KEN BUTLER

PROBLEMS AND SOLUTIONS IN APPLIED STATISTICS

Contents

1
Introduction

This book will hold a collection of problems, and my solutions to them, in applied statistics with R. These come from my courses STAC32 and STAD29 at the University of Toronto Scarborough. I am in the process of adding the latter.

The problems were originally written in Sweave (that is, LaTeX with R code chunks), using the `exam` document class, using data sets stolen from numerous places (textbooks, websites etc). 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.

I just figured out that you can convert LaTeX "label" and "ref" pairs into HTML "a name" and "a href='#' ", which R Markdown can handle. I am ludicrously pleased with myself. To that effect, you will occasionally see question parts beginning with a *; this means that other question parts refer back to this one. (One of my favourite question strategies is to ask how two different approaches lead to the same answer, or more generally to demonstrate that there are different ways to see the same thing.)

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. Likely problems include:

- some LaTeX construction that I didn't catch (eg. block quotes)
- disappeared footnotes (that will show up as an apparently missing sentence in the text)
- references to "in class" or a lecture or a course by course number, which need to be eliminated (in favour of wording like "a previous course")
- references to other questions or question parts that are *wrong*

(likely caused by *not* being "labels" or "refs" in the original LaTeX)
- my contorted English that is difficult to understand.

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

- the STAC32 website
- the STAD29 website

A little background:

STAC32 is an introduction to R (and also SAS) as applied to statistical methods that have (mostly) been learned in previous courses. This could be a mathematical statistics course like this, or a non-mathematical applied course like this. The idea is that students have already seen a little of regression and analysis of variance (and the things that precede them), and need only an introduction of how to run them in R.

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

My checklist:

- bring in the D29 questions
- working on A5 D29
- look for multiple regression Qs in A6 or wherever
- *** puts a vertical line across. Put that above and below each question part, somehow.

2
Getting used to R and R Studio

We begin with this:

```
library(tidyverse)
```

```
## -- Attaching packages ---- tidyverse 1.2.1 --
```

```
## v ggplot2 3.1.0     v purrr   0.2.5
## v tibble  1.4.2     v dplyr   0.7.8
## v tidyr   0.8.1     v stringr 1.3.1
## v readr   1.1.1     v forcats 0.3.0
```

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

and so to the problems:

2.1 *Getting an R Studio Cloud account*

Follow these steps to get an R Studio Cloud account.

(a) Point your web browser at link. (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:

. 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:

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 which `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. 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:

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.

2.2 *Getting started*

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.

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:

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:

(d) Type the line of code shown below into the chunk in the R Note-

book:

```
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.

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).

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
##  * <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>
##  1  21       6  160    110  3.9   2.62  16.5
##  2  21       6  160    110  3.9   2.88  17.0
##  3  22.8     4  108     93  3.85  2.32  18.6
##  4  21.4     6  258    110  3.08  3.22  19.4
##  5  18.7     8  360    175  3.15  3.44  17.0
##  6  18.1     6  225    105  2.76  3.46  20.2
##  7  14.3     8  360    245  3.21  3.57  15.8
##  8  24.4     4  147.    62  3.69  3.19  20
##  9  22.8     4  141.    95  3.92  3.15  22.9
## 10  19.2     6  168.   123  3.92  3.44  18.3
## # ... with 22 more rows, and 4 more
## #   variables: vs <dbl>, am <dbl>,
## #   gear <dbl>, carb <dbl>
```

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:

or the other way:

```r
summary(mtcars)
```

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

```
##  Mean   :3.597   Mean   :3.217
##  3rd Qu.:3.920   3rd Qu.:3.610
##  Max.   :4.930   Max.   :5.424
##       qsec             vs
##  Min.   :14.50   Min.   :0.0000
##  1st Qu.:16.89   1st Qu.:0.0000
##  Median :17.71   Median :0.0000
##  Mean   :17.85   Mean   :0.4375
##  3rd Qu.:18.90   3rd Qu.:1.0000
##  Max.   :22.90   Max.   :1.0000
##        am              gear
##  Min.   :0.0000   Min.   :3.000
##  1st Qu.:0.0000   1st Qu.:3.000
##  Median :0.0000   Median :4.000
##  Mean   :0.4062   Mean   :3.688
##  3rd Qu.:1.0000   3rd Qu.:4.000
##  Max.   :1.0000   Max.   :5.000
##       carb
##  Min.   :1.000
##  1st Qu.:2.000
##  Median :2.000
##  Mean   :2.812
##  3rd Qu.:4.000
##  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:

```
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:

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 Resubmit 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.

2.3 *Reading data from a file*

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

Take a look at the data file at `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:

```
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.

3
Reading in data and drawing some graphs

3.1 *Orange juice*

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

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

Solution
Start with this (almost always):

```
library(tidyverse)
```

```
## -- Attaching packages ---- tidyverse 1.2.1 --
```

```
## v ggplot2 3.1.0     v purrr   0.2.5
## v tibble  1.4.2     v dplyr   0.7.8
## v tidyr   0.8.1     v stringr 1.3.1
## v readr   1.1.1     v forcats 0.3.0
```

```
## -- Conflicts ------- tidyverse_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:

I say *better* because otherwise the read line gets rather long. This way you read it as *the URL is some long thing that I don't care about especially, and I what I need to do is to read the data from that URL, separated by spaces.*

```
url = "http://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: `timinput{.txt}`

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.

3.2 *Making soap*

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

The data are in link.

(a) Read the data into R. Display the data. 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 *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` 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:

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.

```
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:

- create a new data frame that contains everything in `soap` except for `line`

- make a scatter plot of all the points in this new data frame, coloured grey
- plot the points again (from the original data frame), coloured by which production line they're from
- produce a separate scatterplot for each production line.

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
## (Intercept)  80.41099   14.54379   5.529
## speed         1.23074    0.06555  18.775
## lineb       -53.12920    8.21003  -6.471
##             Pr(>|t|)
## (Intercept) 1.10e-05 ***
## speed       7.48e-16 ***
## lineb       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, 7×10^{-16} and 1×10^{-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 *'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: if a variable makes a difference, it should be in your model and on your graph, 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).

Meaning that the graph should contain all three variables, *speed*, *scrap* and *line*.

3.3 *Handling shipments*

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

(a) Read the data into R and display your data frame. 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 *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. That makes sense with our scatterplot, because the *slope* for larger shipments is less than for smaller shipments.

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?

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.

4
Data exploration

```r
library(tidyverse)
```

```
## -- Attaching packages ---- tidyverse 1.2.1 --
```

```
## v ggplot2 3.1.0     v purrr   0.2.5
## v tibble  1.4.2     v dplyr   0.7.8
## v tidyr   0.8.1     v stringr 1.3.1
## v readr   1.1.1     v forcats 0.3.0
```

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

4.1 North Carolina births

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

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

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~
##           <int>        <int>            <int>
##  1           27           26               38
##  2           35           33               40
##  3           34           22               37
##  4           NA           16               38
##  5           35           33               39
##  6           32           24               36
##  7           33           33               38
##  8           38           35               38
##  9           28           29               40
## 10           NA           19               34
## # ... with 490 more rows, and 7 more
## #   variables: ‘Pre-natal Visits‘ <int>,
## #   ‘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, 3...
## $ ‘Mother Age‘           <int> 26, 33, 2...
## $ ‘Weeks Gestation‘      <int> 38, 40, 3...
## $ ‘Pre-natal Visits‘     <int> 14, 11, 1...
## $ ‘Marital Status‘       <int> 1, 1, 2, ...
## $ ‘Mother Weight Gained‘ <int> 32, 23, 5...
## $ ‘Low Birthweight?‘     <int> 0, 0, 0, ...
## $ ‘Weight (pounds)‘      <dbl> 6.8750, 6...
## $ ‘Premie?‘              <int> 0, 0, 0, ...
## $ ‘Few Visits?‘          <int> 0, 0, 0, ...
```

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

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 have, and the first few of the values.

The variable `Weight (pounds)` is the birthweight (in pounds), `Premie?` is 1 for a premature baby and 0 for a full-term baby, and `Weeks Gestation` is the number of weeks the pregnancy lasted. Don't forget to put backticks around each of those when you use them later.

these are mostly *int* or *integer*.

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

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: link. The nice thing about Sturges' rule, mentioned there, is that you can almost get a number of bins for your histogram in your head (as long

as you know the powers of 2, that is). What you do is to start with your sample size, here $n = 500$. You find the next power of 2 above that, which is here $512 = 2^9$. You then take that power and add 1, to get 10 bins. If you don't like that, you can get R to calculate it for you:

```
nclass.Sturges(bw$`Weight (pounds)`)
```

```
## [1] 10
```

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

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

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

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

```
w = 2 * IQR(bw$`Weight (pounds)`) * 500^(-1/3)
w
```

```
## [1] 0.4094743
```

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

R also has

```
nc = nclass.FD(bw$'Weight (pounds)')
nc
```

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

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

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

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

```
ggplot(bw, aes(x = 'Weight (pounds)')) + geom_density()
```

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

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

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

If you think this is approximately normal, you need to make some comment along the lines of "the shape is approximately symmetric with no outliers". I think my first answer is better, but this answer is worth something, since it is a not completely unreasonable interpretation of the histogram.

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

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

The high weight is actually an outlier, but look at all those outliers at the bottom!

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.

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.

4.2 *More about the NC births*

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

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

Solution

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

```
ggplot(bw, aes(x = factor(`Premie?`), y = `Weeks Gestation`)) +
    geom_boxplot()
```

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

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

```
ggplot(bw, aes(x = `Premie?`, y = `Weeks Gestation`)) +
    geom_point()
```

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

The same warning again, for the same reason.

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

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

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

```
bw %>% summarize(n = n(), min = min('Weeks Gestation'),
    max = max('Weeks Gestation'))
```

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

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

I explain the missing values below.

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

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.

4.3 *Nenana, Alaska*

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

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

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

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          expected   <int> <chr> <chr>
## ... ................. ... ........................................... ........ ......................
## 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 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 col~ 'http://www~
##  2     2 <NA>  1 columns 5 col~ 'http://www~
##  3     3 <NA>  1 columns 5 col~ 'http://www~
##  4     4 <NA>  1 columns 5 col~ 'http://www~
##  5     5 <NA>  1 columns 5 col~ 'http://www~
##  6     6 <NA>  1 columns 5 col~ 'http://www~
##  7     7 <NA>  1 columns 5 col~ 'http://www~
##  8     8 <NA>  1 columns 5 col~ 'http://www~
##  9     9 <NA>  1 columns 5 col~ 'http://www~
## 10    10 <NA>  1 columns 5 col~ 'http://www~
## # ... 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, 19...
## $ JulianDate  <dbl> 120.4795, 131.3983, ...
## $ ‘Date&Time‘ <chr> "April 30 at 11:30 A...
```

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.

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
##    <int>      <dbl> <chr>       <chr>
##  1  1917       120. April 30 a~ 1917 ~
##  2  1918       131. May 11 at ~ 1918 ~
##  3  1919       124. May 3 at 2~ 1919 ~
##  4  1920       132. May 11 at ~ 1920 ~
##  5  1921       131. May 11 at ~ 1921 ~
##  6  1922       133. May 12 at ~ 1922 ~
##  7  1923       129. May 9 at 2~ 1923 ~
##  8  1924       133. May 11 at ~ 1924 ~
##  9  1925       128. May 7 at 6~ 1925 ~
## 10  1926       117. April 26 a~ 1926 ~
## # ... with 77 more rows, and 1 more
## #   variable: datetime <dttm>
```

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

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

```
nenana2 <- nenana %>% mutate(longdt = str_c(Year,
    " ", `Date&Time`)) %>% mutate(datetime = ymd_hm(longdt,
```

```
    tz = "America/Anchorage")) %>% mutate(jd = yday(datetime))
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. `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 `lubridate`. What has actually happened is a quirk of how tibbles are displayed: they show 3 significant digits, *rounded*. The Julian dates given by `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:

```
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: `jd` is (to this accuracy) a whole number, but `JulianDate` is a decimal with fractional part greater than 0.50.

Now I have to turn the extra signficant digits off:

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

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

Solution

`geom_point`:

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

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

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

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

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

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

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

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

4.4 *Computerized accounting*

Beginning accounting students need to learn to learn to audit in a computerized environment. A sample of beginning accounting students took each of two tests: the Computer Attitude Scale (CAS) and the Computer Anxiety Rating Scale (CARS). A higher score in

each indicates greater anxiety around computers. The test scores are scaled to be between 0 and 5. Also noted was each student's gender. The data are in `http://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
##  Length:35          Min.   :1.800
##  Class :character   1st Qu.:2.575
##  Mode  :character   Median :2.800
##                     Mean   :2.816
##                     3rd Qu.:3.150
##                     Max.   :3.750
##       CARS
##  Min.   :1.000
##  1st Qu.:2.445
##  Median :2.790
##  Mean   :2.771
##  3rd Qu.:3.290
##  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.

4.5 *Test scores in two classes*

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

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

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

(a) Read the data into a data frame called `marks`, using `read_delim`, and list the data frame (by typing its name) to confirm that you

read the data values properly. Note that the top line of the data file contains the names of the variables, as it ought to.

Solution

I was lazy and used the one on the web, the values being separated ("delimited") by exactly one space:

```
my_url = "http://www.utsc.utoronto.ca/~butler/d29/marks.txt"
marks = read_delim(my_url, " ")
```

```
## Parsed with column specification:
## cols(
##   class = col_character(),
##   score = col_integer()
## )
```

```
marks
```

```
## # A tibble: 11 x 2
##    class  score
##    <chr>  <int>
##  1 ken       78
##  2 ken       62
##  3 ken       59
##  4 ken       69
##  5 ken       81
##  6 thomas    83
##  7 thomas    77
##  8 thomas    63
##  9 thomas    61
## 10 thomas    79
## 11 thomas    72
```

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

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

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

```
## Parsed with column specification:
## cols(
##   X1 = col_character(),
```

```
##   X2 = col_integer()
## )

## # A tibble: 11 x 2
##    X1        X2
##    <chr>  <int>
##  1 ken       78
##  2 ken       62
##  3 ken       59
##  4 ken       69
##  5 ken       81
##  6 thomas    83
##  7 thomas    77
##  8 thomas    63
##  9 thomas    61
## 10 thomas    79
## 11 thomas    72
```

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

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

## Parsed with column specification:
## cols(
##   instructor = col_character(),
##   mark = col_integer()
## )

## # A tibble: 11 x 2
##    instructor  mark
##    <chr>      <int>
##  1 ken           78
##  2 ken           62
##  3 ken           59
##  4 ken           69
##  5 ken           81
##  6 thomas        83
##  7 thomas        77
##  8 thomas        63
##  9 thomas        61
## 10 thomas        79
## 11 thomas        72
```

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

Solution

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

Remember: on a regular boxplot, the groups go across (x), the variable measured goes up (y).

Boxplots can also go across the page, but for us, they don't.

Extra: this might work:

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

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

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

Solution

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

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

Solution

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

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

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

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

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

Solution

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

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

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

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

```
## [1] 72
```

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

```
## [1] 72
```

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

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

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

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

```
## # A tibble: 2 x 2
##   class    med
```

```
##   <chr>  <dbl>
## 1 ken     69
## 2 thomas  74.5
```

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

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

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

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

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

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

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

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

```
marks %>% nest(-class)
```

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

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

Now, for each of those mini-data-frames, we want to calculate the quantiles of `score`. To do that, use `map`, which is the `tidyverse` version of "for each": for each of our mini-data-frames, calculate the five-number summary of the column called `score` in *it*:

```
marks %>% nest(-class) %>% mutate(qq = map(data,
    ~quantile(.$score)))
```

```
## # A tibble: 2 x 3
##   class  data             qq
##   <chr>  <list>           <list>
## 1 ken    <tibble [5 x 1]> <dbl [5]>
## 2 thomas <tibble [6 x 1]> <dbl [5]>
```

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

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

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

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

```
marks %>% nest(-class) %>% mutate(qq = map(data,
    ~quantile(.$score))) %>% unnest(qq)
```

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

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

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

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

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

or thus:

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

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

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

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

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

```
marks %>% nest(-class) %>% mutate(qq = map(data,
    ~enframe((quantile(.$score)))))
```

```
## # A tibble: 2 x 3
##   class  data             qq
##   <chr>  <list>           <list>
## 1 ken    <tibble [5 x 1]> <tibble [5 x 2]>
## 2 thomas <tibble [6 x 1]> <tibble [5 x 2]>
```

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

And finally `unnest qq`:

```
marks %>% nest(-class) %>% mutate(qq = map(data,
    ~enframe((quantile(.$score))))) %>% unnest(qq)
```

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

Success! Or even:

```
marks %>% nest(-class) %>% mutate(qq = map(data,
    ~enframe((quantile(.$score))))) %>% unnest(qq) %>%
    mutate(qn = parse_number(name)) %>% select(-name) %>%
    spread(qn, value)
```

```
## # A tibble: 2 x 6
##   class    `0`  `25`  `50`  `75` `100`
##   <chr>  <dbl> <dbl> <dbl> <dbl> <dbl>
## 1 ken       59  62    69    78      81
## 2 thomas    61  65.2  74.5  78.5    83
```

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

5
One-sample inference

```r
library(tidyverse)
```

```
## -- Attaching packages ---- tidyverse 1.2.1 --
```

```
## v ggplot2 3.1.0     v purrr   0.2.5
## v tibble  1.4.2     v dplyr   0.7.8
## v tidyr   0.8.1     v stringr 1.3.1
## v readr   1.1.1     v forcats 0.3.0
```

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

5.1 Hunter-gatherers in Australia

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 μ 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 α 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.

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.

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.

5.2 *Buses to Boulder*

A bus line operates a route from Denver to Boulder (these places are in Colorado). The schedule says that the journey time should be 60 minutes. 11 randomly chosen journey times were recorded, and these are in the file link, with journey times shown in minutes.

(a) Read the data into R, and display the data frame that you read in.

Solution

Since you can read the data directly from the URL, do that (if you are online) rather than having to copy and paste and save, and then find the file you saved. Also, there is only one column, so you can pretend that there were multiple columns, separated by whatever you like. It's least typing to pretend that they were separated by commas like a `.csv` file:

```
my_url = "http://www.utsc.utoronto.ca/~butler/d29/buses.txt"
journey.times = read_csv(my_url)
```

```
## Parsed with column specification:
## cols(
##   minutes = col_integer()
## )
```

```
journey.times
```

```
## # A tibble: 11 x 1
##    minutes
##      <int>
##  1      58
##  2      61
##  3      69
```

```
##  4      62
##  5      81
##  6      54
##  7      72
##  8      71
##  9      53
## 10      54
## 11      66
```

Using `read_delim` with any delimiter (such as `" "`) will also work, and is thus also good.

Variable names in R can have a dot (or an underscore, but not a space) in them. I have grown accustomed to using dots to separate words. This works in R but not other languages, but is seen by some as old-fashioned, with underscores being the modern way. You can also use what is called "camel case" by starting each "word" after the first with an uppercase letter like this:

In some languages, a dot is used to concatenate bits of text, or as a way of calling a method on an object. But in R, a dot has no special meaning, and is used in function names like *t.test*. Or *read.table*.

```
journeyTimes = read_csv(my_url)
```

You have to get the capitalization and punctuation right when you use your variables, no matter what they're called. In any of the cases above, there is no variable called `journeytimes`. As Jenny Bryan (in link) puts it, boldface in original: Implicit contract with the computer / scripting language: Computer will do tedious computation for you. In return, you will be completely precise in your instructions. Typos matter. Case matters. **Get better at typing.**

(b) Run a suitable test to see whether there is evidence that the mean journey time differs from 60 minutes. What do you conclude? (I want a conclusion that says something about journey times of buses.)

Solution

`t.test` doesn't take a `data=` to say which data frame to use. Wrap it in a `with`:

```
with(journey.times, t.test(minutes, mu = 60))
```

```
##
##  One Sample t-test
##
## data:  minutes
## t = 1.382, df = 10, p-value = 0.1971
## alternative hypothesis: true mean is not equal to 60
## 95 percent confidence interval:
```

```
##  57.71775 69.73680
## sample estimates:
## mean of x
##  63.72727
```

We are testing that the mean journey time is 60 minutes, against the two-sided alternative (default) that the mean is not equal to 60 minutes. The P-value, 0.1971, is a lot bigger than the usual α of 0.05, so we cannot reject the null hypothesis. That is, there is no evidence that the mean journey time differs from 60 minutes.

As you remember, we have not proved that the mean journey time *is* 60 minutes, which is what "accepting the null hypothesis" would be. We have only failed to reject it, in a shoulder-shrugging kind of way: "the mean journey time *could* be 60 minutes". The other acceptable word is "retain"; when you say "we retain the null hypothesis", you imply something like "we act as if the mean is 60 minutes, at least until we find something better."

(c) Give a 95% confidence interval for the mean journey time. (No R code is needed here.)

Solution

Just read it off from the output: 57.72 to 69.74 minutes.

(d) Do you draw consistent conclusions from your test and confidence interval? Explain briefly.

Solution

The test said that we should not reject a mean of 60 minutes. The confidence interval says that 60 minutes is inside the interval of plausible values for the population mean, which is another way of saying the same thing. (If we had rejected 60 as a mean, 60 would have been *outside* the confidence interval.)

(e) Draw a boxplot of the journey times. Do you see a reason to doubt the test that you did above?

Solution

The grouping variable is a "nothing" as in the Ken and Thomas question (part (d)):

```
ggplot(journey.times, aes(x = 1, y = minutes)) +
    geom_boxplot()
```

The assumption behind the *t*-test is that the population from which the data come has a normal distribution: ie. symmetric with no outliers. A small sample (here we have 11 values) even from a normal distribution might look quite non-normal (as in Assignment 0 from last week), so I am not hugely concerned by this boxplot. However, it's perfectly all right to say that this distribution is skewed, and therefore we should doubt the *t*-test, because the upper whisker is longer than the lower one. In fact, the topmost value is very nearly an outlier:

```
ggplot(journey.times, aes(x = minutes)) + geom_histogram(bins = 5)
```

Whether you think it is or not may depend on how many bins you have on your histogram. With 5 bins it looks like an outlier, but with 6 it does not. Try it and see.

and there might be skewness as well, so maybe I should have been concerned.

I would be looking for some intelligent comment on the boxplot: what it looks like vs. what it ought to look like. I don't so much mind what that comment is, as long as it's intelligent enough.

Perhaps I should draw a normal quantile plot:

```
ggplot(journey.times, aes(sample = minutes)) +
    stat_qq() + stat_qq_line()
```

The normal quantile plot is saying that the problem is actually at the *bottom* of the distribution: the lowest value is not low enough, but the highest value is actually *not* too high. So this one seems to be on the edge between OK and being right-skewed (too bunched up at the bottom). My take is that with this small sample this is not too bad. But you are free to disagree.

If you don't like the normality, you'd use a *sign test* and test that the *median* is not 60 minutes, which you would (at my guess) utterly fail to reject:

```
library(smmr)
sign_test(journey.times, minutes, 60)

## $above_below
## below above
##     4     7
##
## $p_values
##   alternative   p_value
## 1       lower 0.8867187
## 2       upper 0.2744141
## 3   two-sided 0.5488281

ci_median(journey.times, minutes)

## [1] 54.00195 71.99023
```

and so we do. The median could easily be 60 minutes.

5.3 *Length of gestation in North Carolina*

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

Let μ be the population mean (the mean weight of all babies born in North Carolina). Null hypothesis is $H_0 : \mu = 7.3$ pounds, and the alternative is that the mean is less: $H_a : \mu < 7.3$ pounds.

Note that I defined μ first before I used it.

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 α 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.

5.4 *Inferring ice break-up in Nenana*

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

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

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 = 0.9))
```

```
##
##  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:

```
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 α 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:

```
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.

6
Two-sample inference

```
library(tidyverse)
```

```
## -- Attaching packages ---- tidyverse 1.2.1 --
```

```
## v ggplot2 3.1.0     v purrr   0.2.5
## v tibble  1.4.2     v dplyr   0.7.8
## v tidyr   0.8.1     v stringr 1.3.1
## v readr   1.1.1     v forcats 0.3.0
```

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

6.1 Children and electronic devices

Do children (aged 8–17) spend more time on electronic devices now than they did 10 years ago? Samples of 15 children aged 8–17 were taken in each of two years, 1999 and 2009, and the children (with their parents' help) were asked to keep a diary of the number of hours they spent using electronic devices on a certain day. The data are in the file `http://www.utsc.utoronto.ca/~butler/c32/pluggedin.txt`.

(a) Read in the data and verify that you have 30 rows of data from two different years.

Solution
I see this:

```
myurl = "http://www.utsc.utoronto.ca/~butler/c32/pluggedin.txt"
plugged = read_delim(myurl, " ")
```

```
## Parsed with column specification:
## cols(
```

```
##   year = col_integer(),
##   hours = col_integer()
## )
```

```
plugged
```

```
## # A tibble: 30 x 2
##     year hours
##    <int> <int>
##  1  1999     4
##  2  1999     5
##  3  1999     7
##  4  1999     7
##  5  1999     5
##  6  1999     7
##  7  1999     5
##  8  1999     6
##  9  1999     5
## 10  1999     6
## # ... with 20 more rows
```

I see only the first ten rows (with an indication that there are 20 more, so 30 altogether). In your notebook, it'll look a bit different: again, you'll see the first 10 rows, but you'll see exactly how many rows and columns there are, and there will be buttons "Next" and "Previous" to see earlier and later rows, and a little right-arrow to see more columns to the right (to which is added a little left-arrow if there are previous columns to scroll back to). If you want to check for yourself that there are 30 rows, you can click Next a couple of times to get down to row 30, and then see that the Next button cannot be clicked again, and therefore that 30 rows is how many there are.

Or, you can summarize the years by counting how many there are of each:

```
plugged %>% count(year)
```

```
## # A tibble: 2 x 2
##    year     n
##   <int> <int>
## 1  1999    15
## 2  2009    15
```

or the more verbose form of the same thing:

```
plugged %>% group_by(year) %>% summarize(rows = n())
```

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

```
##    year  rows
##   <int> <int>
## 1  1999    15
## 2  2009    15
```

Any of those says that it looks good. 30 rows, 1999 and 2009, 15 measurements for each.

(b) Draw side-by-side boxplots of the number of hours for each year. `year` is a numeric variable that we want to treat as a factor, so we need to *make* it into a factor.

Solution

```
ggplot(plugged, aes(x = factor(year), y = hours)) +
    geom_boxplot()
```

The `fct_inorder` trick from assignment 1 will also work, since the years are in the data in the order we want them to be displayed.

The median for 2009 is noticeably higher, and there is no skewness or outliers worth worrying about.

The measurements for the two years have a very similar spread, so there would be no problem running the pooled test here.

You might be bothered by the `factor(year)` on the x-axis. To get around that, you can define year-as-factor *first*, using `mutate`, then feed your new column into the boxplot. That goes like this. There is a wrinkle that I explain afterwards:

```
plugged %>% mutate(the_year = factor(year)) %>%
    ggplot(aes(x = the_year, y = hours)) + geom_boxplot()
```

You could even redefine `year` to be the factor version of itself (if you don't need the year-as-number anywhere else). The wrinkle I mentioned above is that in the `ggplot` you *do not* name the data frame first; the data frame used is the (nameless) data frame that came out of the previous step, not `plugged` but `plugged` with a new column `the_year`.

Note how the *x*-axis now has the name of the new variable.

If you forget to make `year` into a factor, this happens:

```
ggplot(plugged, aes(x = year, y = hours)) + geom_boxplot()
```

```
## Warning: Continuous x aesthetic -- did you
## forget aes(group=...)?
```

You get *one* boxplot, for all the hours, without distinguishing by year, and a warning message that tries (and fails) to read our mind: yes, we have a continuous, quantitative `x`, but `geom_boxplot` doesn't take a `group`.

(c) Test whether the mean number of hours has *increased* since

1999. Which test did R do?

Solution

The hard part to remember is how you specify a one-sided test in R; it's `alternative="less"` (rather than "greater") because 1999 is "before" 2009:

```
t.test(hours ~ year, data = plugged, alternative = "less")
```

```
##
##  Welch Two Sample t-test
##
## data:  hours by year
## t = -3.3323, df = 24.861, p-value =
## 0.001348
## alternative hypothesis: true difference in means is less than 0
## 95 percent confidence interval:
##        -Inf -0.8121415
## sample estimates:
## mean in group 1999 mean in group 2009
##           5.933333           7.600000
```

The P-value is 0.0013. R does the Welch-Satterthwaite test by default (the unequal-variances one). Since we didn't change that, that's what we got. (The pooled test is below.)

This is the cleanest way to do it, because this version of `t.test`, with a "model formula" (the thing with the squiggle) allows a `data=` to say which data frame to get things from. The other ways, using (for example) `with`, also work:

```
with(plugged, t.test(hours ~ year, alternative = "less"))
```

```
##
##  Welch Two Sample t-test
##
## data:  hours by year
## t = -3.3323, df = 24.861, p-value =
## 0.001348
## alternative hypothesis: true difference in means is less than 0
## 95 percent confidence interval:
##        -Inf -0.8121415
## sample estimates:
## mean in group 1999 mean in group 2009
##           5.933333           7.600000
```

This also works, but is *ugly*:

```
t.test(plugged$hours ~ plugged$year, alternative = "less")
```

```
##
##  Welch Two Sample t-test
##
## data:  plugged$hours by plugged$year
## t = -3.3323, df = 24.861, p-value =
## 0.001348
## alternative hypothesis: true difference in means is less than 0
## 95 percent confidence interval:
##        -Inf -0.8121415
## sample estimates:
## mean in group 1999 mean in group 2009
##           5.933333           7.600000
```

Ugly because you've just typed the name of the data frame and the dollar sign *twice* for no reason. As a general principle, if you as a programmer are repeating yourself, you should stop and ask yourself how you can avoid the repeat.

If you want the pooled test in R, you have to ask for it:

```
t.test(hours ~ year, alternative = "less", data = plugged,
    var.equal = T)
```

```
##
##  Two Sample t-test
##
## data:  hours by year
## t = -3.3323, df = 28, p-value =
## 0.001216
## alternative hypothesis: true difference in means is less than 0
## 95 percent confidence interval:
##        -Inf -0.8158312
## sample estimates:
## mean in group 1999 mean in group 2009
##           5.933333           7.600000
```

As is often the case, the P-values for the pooled and Welch-Satterthwaite tests are very similar, so from that point of view it doesn't matter much which one you use. If you remember back to the boxplots, the number of hours had about the same spread for the two years, so if you used the pooled test instead of the Welch-Satterthwaite test, that would have been just fine.

There is a school of thought that says we should learn the Welch-Satterthwaite test and use that always. This is because W-S (i) works

when the populations from which the groups are sampled have different SDs and (ii) is pretty good even when those SDs are the same.

The pooled test can go badly wrong if the groups have very different SDs. The story is this: if the larger sample is from the population with the larger SD, the probability of a type I error will be smaller than α, and if the larger sample is from the population with the *smaller* SD, the probability of a type I error will be larger than α. This is why you see S-W in STAB22. You see the pooled test in STAB57 because the logic of its derivation is so much clearer, not because it's really the better test in practice. The theory says that if your data are normal in both groups with the same variance, then the pooled test is best, but it says *nothing* about the quality of the pooled test if any of that goes wrong. The usual approach to assessing things like this is via simulation, as we do for estimating power (later): generate some random data eg. from normal distributions with the same means, SDs 10 and 20 and sample sizes 15 and 30, run the pooled t-test, see if you reject, then repeat lots of times and see whether you reject about 5% of the time. Then do the same thing again with the sample sizes switched around. Or, do the same thing with Welch-Satterthwaite.

I return to this issue when we look at the same data in SAS later.

(d) Obtain a 99% confidence interval for the difference in means.

Solution

Take off the thing that made it one-sided, and put in a thing that gets the right CI:

```
t.test(hours ~ year, data = plugged, conf.level = 0.99)
```

```
##
##  Welch Two Sample t-test
##
## data:  hours by year
## t = -3.3323, df = 24.861, p-value =
## 0.002696
## alternative hypothesis: true difference in means is not equal to 0
## 99 percent confidence interval:
##  -3.0614628 -0.2718705
## sample estimates:
## mean in group 1999 mean in group 2009
##           5.933333           7.600000
```

-3.06 to -0.27. The interval contains only negative values, which is consistent with our having rejected a null hypothesis of no difference in means.

6.2 *Parking close to the curb*

In 2009, the Toronto Star commissioned a survey to address the issue of who is better at parking a car: men or women. The researchers recorded 93 drivers who were parallel-parking their car in downtown Toronto, and for each driver, recorded the distance between the car and the curb, in inches, when the driver was finished parking their car. The data are in an Excel spreadsheet, link. (Let me know if you cannot handle this format.) Click on the link. The data will probably download automatically. Check the folder on your computer where things get downloaded. If the spreadsheet is just displayed and not downloaded, save it somewhere on your computer.

Mine is rather prosaically called *Downloads*.

(a) There are two sheets in this spreadsheet workbook. They are of the same data in two different formats. Take a look at Sheet 1 and Sheet 2. Describe the format of the data in each case. Which will be the most suitable data layout, bearing in mind that one of the first things we do is to make side-by-side boxplots of parking distances for males and females? Explain briefly.

Solution

The data in Sheet 1 has one column of parking distances for males, and another for females. This is often how you see data of this sort laid out. Sheet 2 has *one* column of parking distances, all combined together, and a second column indicating the gender of the driver whose distance is in the first column. If you look back at the kind of data we've used to make side-by-side boxplots, it's always been in the format of Sheet 2: one column containing all the values of the variable we're interested in, with a second column indicating which group each observation belongs to ("group" here being "gender of driver"). So we need to use the data in Sheet 2, because the data in Sheet 1 are not easy to handle with R. The layout of Sheet 2 is the way R likes to do most things: so-called "long format" with a lot of rows and not many columns. This is true for descriptive stuff: side-by-side boxplots or histograms or means by group, as well as modelling such as (here) a two-sample t-test, or (in other circumstances, as with several groups) a one-way analysis of variance. Hadley Wickham, the guy behind the `tidyverse`, likes to talk about "tidy data" (like Sheet 2), with each column containing a variable, and "untidy data" (like Sheet 1), where the two columns are the same thing (distances), but under different circumstances (genders). As we'll see later, it is possible to convert from one format to the other. Usually you want to make untidy data tidy (the function for this is called `gather`).

(b) Read your preferred sheet directly into R, *without* using a `.csv`

file. (There is a clue in the lecture notes, in the section about reading in files.) If you get stuck, make a `.csv` file and read that in.

Solution

The direct way is to use the package `readxl`. This has a `read_excel` that works the same way as any of the other `read_` functions. You'll have to make sure that you read in sheet `2`, since that's the one you want. There is some setup first. There are a couple of ways you can do that:

- Download the spreadsheet to your computer, and upload it to your project on R Studio Cloud (or, if you are running R Studio on your computer, use something like `file.choose` to get the file from wherever it got downloaded to).
- Use the function `download.file` to get the file from the URL and store it in your project folder directly. This also works in R Studio Cloud, and completely by-passes the download-upload steps that you would have to do otherwise. (I am grateful to Rose Gao for this idea.) Here is how you can use `download.file` here:

```
my_url = "http://www.utsc.utoronto.ca/~butler/c32/parking.xlsx"
local = "parking.xlsx"
download.file(my_url, local, mode = "wb")
```

When you've gotten the spreadsheet into your project folder via one of those two ways, you go ahead and do this:

```
library(readxl)
parking = read_excel("parking.xlsx", sheet = 2)
parking
```

```
## # A tibble: 93 x 2
##    distance gender
##       <dbl> <chr>
##  1      0.5 male
##  2      1   male
##  3      1.5 male
##  4      1.5 male
##  5      1.5 male
##  6      3   male
##  7      3.5 male
##  8      5   male
##  9      6   male
## 10      6   male
## # ... with 83 more rows
```

You have to do it this way, using the version of the spreadsheet on your computer, since `read_excel` won't take a URL, or if it does, I can't make it work. I put the spreadsheet in R Studio's current folder, so I could read it in by name, or you can do the `f=file.choose()` thing, find it, then read it in. The `sheet=` thing can take either a number (as here: the second sheet in the workbook), or a name (whatever name the sheet has on its tab in the workbook).

Let me know if you have more success than I did.

Extra: Rose actually came up with a better idea, which I will show you and explain:

```
tf = tempfile()
download.file(my_url, tf, mode = "wb")
p = read_excel(tf, sheet = 2)
```

What `tempfile()` does is to create a temporary file to hold the spreadsheet that you are about to download. After downloading the spreadsheet to the temporary file, you then use `read_excel` to read *from the temporary file* into the data frame.

The advantage of this approach is that the temporary file disappears as soon as you close R, and so you don't have a copy of the spreadsheet lying around that you don't need (once you have created the data frame that I called `parking`, anyway).

If you are wondering about that `mode` thing on `download.file`: files are of two different types, "text" (like the text of an email, that you can open and look at in something like Notepad), and "binary" that you can't look at directly, but for which you need special software like Word or Excel to decode it for you.

A Word or Excel document has all kinds of formatting information hidden in the file as well as the text that you see on the screen.

The first character in `mode` is either `w` for "write a new file", which is what we want here, or `a` for "append", which would mean adding to the end of a file that already exists. Thus `mode="wb"` means "create a new binary file". End of Extra. If you can't make any of this work, then do it in two steps: save the appropriate sheet as a `.csv` file, and then read the `.csv` file using `read_csv`. If you experiment, you'll find that saving a spreadsheet workbook as `.csv` only saves the sheet you're looking at, so make sure you are looking at sheet `2` before you Save As `.csv`. I did that, and called my saved `.csv` `parking2.csv` (because it was from sheet `2`, but you can use any name you like). Then I read this into R thus:

```
parking2 = read_csv("parking2.csv")
```

```
## Parsed with column specification:
## cols(
##   distance = col_double(),
##   gender = col_character()
## )
```

```
parking2
```

```
## # A tibble: 93 x 2
##    distance gender
##       <dbl> <chr>
##  1      0.5 male
##  2      1   male
##  3      1.5 male
##  4      1.5 male
##  5      1.5 male
##  6      3   male
##  7      3.5 male
##  8      5   male
##  9      6   male
## 10      6   male
## # ... with 83 more rows
```

The read-in data frame `parking` has 93 rows ($47 + 46 = 93$ drivers) and two columns: the distance from the curb that the driver ended up at, and the gender of the driver. This is as the spreadsheet Sheet 2 was, and the first few distances match the ones in the spreadsheet.

If I were grading this, you'd get some credit for the `.csv` route, but I really wanted you to figure out how to read the Excel spreadsheet directly, so that's what would be worth full marks.

You might want to check that you have some males and some females, and how many of each, which you could do this way:

```
parking %>% count(gender)
```

```
## # A tibble: 2 x 2
##   gender     n
##   <chr>  <int>
## 1 female    47
## 2 male      46
```

(c) Obtain side-by-side boxplots of parking distances for males and females. Does one gender seem to be better at parking than the other? Explain briefly.

Solution

With the right data set, this is a piece of cake:

```
ggplot(parking, aes(x = gender, y = distance)) +
    geom_boxplot()
```

The outcome variable is distance from the curb, so smaller should be better (more accurate parking). With that in mind, the median for females is a little smaller than for males (about 8.5 vs. about 10), so it seems that on average females are more accurate parkers than males are. The difference is small, however (and so you might be wondering at this point whether it's a statistically significant difference — don't worry, that's coming up).

Before I leave this one, I want to show you something else: above-and-below histograms, as another way of comparing males and females (two or more groups, in general). First, we make a histogram of all the distances, without distinguishing by gender:

```
ggplot(parking, aes(x = distance)) + geom_histogram(bins = 8)
```

That big outlier is the very inaccurate male driver.

Now, how do we get a *separate* histogram for each `gender`? In `ggplot`, separate plots for each "something" are called **facets**, and the

way to get facets arranged as you want them is called `facet_grid`. Let me show you the code first, and then explain how it works:

```
ggplot(parking, aes(x = distance)) + geom_histogram(bins = 7) +
    facet_grid(gender ~ .)
```

`facet_grid` takes a "model formula" with a squiggle, with y on the left and x on the right. We want to compare our two histograms, one for males and one for females, and I think the best way to compare histograms is to have one on top of the other. Note that the same `distance` scale is used for both histograms, so that it is a fair comparison. The above-and-below is accomplished by having `gender` as the y in the arrangement of the facets, so it goes before the squiggle. We don't have any x in the arrangement of the facets, and we tell `ggplot` this by putting a dot where the x would be.

You might have a second categorical variable by which you want to arrange the facets left and right, and that would go where the dot is.

You can also use `facet_wrap` for this, but you have to be more careful since you don't have any control over how the histograms come out (you probably get them side by side, which is not so helpful for comparing distributions). You can make it work by using `ncol=1` to arrange "all" the histograms in one column:

```
ggplot(parking, aes(x = distance)) + geom_histogram(bins = 7) +
    facet_wrap(~gender, ncol = 1)
```

The centres of both histograms are somewhere around 10, so it's hard to see any real difference between males and females here. Maybe this is further evidence that the small difference we saw between the boxplots is really not worth getting excited about.

You might be concerned about how you know what to put with the squiggle-thing in `facet_grid` and `facet_wrap`. The answer is that `facet_wrap` *only* has something to the right of the squiggle (which `ggplot` then decides how to arrange), but `facet_grid` *must* have something on *both* sides of the squiggle (how to arrange in the y direction on the left, how to arrange in the x direction on the right), and if you don't have anything else to put there, you put a dot. Here's my `facet_grid` code from above, again:

```
ggplot(parking, aes(x = distance)) + geom_histogram(bins = 7) +
    facet_grid(gender ~ .)
```

We wanted gender to go up and down, and we had nothing to go left and right, hence the dot. Contrast that with my `facet_wrap` code:

I took out the *ncol* since that confuses the explanation here.

```
ggplot(parking, aes(x = distance)) + geom_histogram(bins = 7) +
    facet_wrap(~gender)
```

This says "make a separate facet for each gender", but it doesn't say anything about how to arrange them. The choice of bins for my histogram(s) came from Sturges' rule: with n being the number of observations, you use k bins where $k = \log_2(n) + 1$, rounded up. If we were to make a histogram of all the parking distances combined together, we would have $n = 47 + 48 = 95$ observations, so we should use this many bins:

```
sturges = log(95, 2) + 1
sturges
```

```
## [1] 7.569856
```

Round this up to 8. (The second thing in `log` is the base of the logs, if you specify it, otherwise it defaults to e and gives you "natural" logs.) I seem to have the powers of 2 in my head, so I can do it mentally by saying "the next power of 2 is 128, which is 2^7, so I need $7 + 1 = 8$ bins."

Or:

```
with(parking, nclass.Sturges(distance))
```

```
## [1] 8
```

Sturges' rule tends to produce not enough bins if n is small, so be prepared to increase it a bit if you don't have much data. I think that gives a fairly bare-bones picture of the shape: skewed to the right with outlier.

The other rule we saw was Freedman-Diaconis:

```
with(parking, nclass.FD(distance))
```

```
## [1] 14
```

and that leads to this histogram:

```
ggplot(parking, aes(x = distance)) + geom_histogram(bins = 14)
```

That gives rather more detail (a lot more bars: the binwidth in the Sturges-rule histogram is about 7, or twice what you see here), but in this case the overall story is about the same.

In the case of faceted histograms, you would want to apply a rule that uses the number of observations in *each* histogram. The facets might have quite different numbers of observations, but you can only

use one `binwidth` (or `bins`), so you may have to compromise. For example, using Sturges' rule based on 47 observations (the number of males; the number of females is one more):

```
log(47, 2) + 1
```

```
## [1] 6.554589
```

and so each facet should have that many bins, rounded up. That's where I got my 7 for the facetted histogram from. This one doesn't work immediately with `nclass.Sturges`, because we do not have *one* column whose length is the number of observations we want: we have one column of distances that are males and females mixed up. To do *that*, `filter` one of the genders first:

```
parking %>% filter(gender == "female") %>% with(.,
    nclass.Sturges(distance))
```

```
## [1] 7
```

I used the "dot" trick again, which you can read as "it": "from `parking`, take only the rows for the females, and with it, give me the number of bins for a histogram by Sturges' rule."

(d) Explain briefly why this is two independent samples rather than matched pairs.

Solution

There is no way to pair any male with a corresponding female, because they are unrelated people. You might also notice that there are not even the *same number* of males and females, so there can be no way of pairing them up without leaving one over. (In general, if the two samples are paired, there must be the same number of observations in each; if there are different numbers in each, as here, they cannot be paired.) If you want that more mathematically, let n_1 and n_2 be the two sample sizes; then:

$$\text{Paired} \Longrightarrow n_1 = n_2$$

from which it follows logically (the "contrapositive") that

$$n_1 \neq n_2 \Longrightarrow \text{not paired}$$

You'll note from the logic that if the two sample sizes are the same, that tells you *nothing* about whether it's paired or independent samples: it could be either, and in that case you have to look at the description of the data to decide between them.

Here, anything that gets at why the males and females cannot be paired up is good.

(e) Run a suitable t-test for comparing parking distances for males and females. What do you conclude, in the context of the data?

Solution

A two-sample t-test. I think either the Welch or the pooled one can be justified (and I would expect them to give similar answers). You can do the Welch one either without comment or by asserting that the boxplots show different spreads; if you are going to do the pooled one, you need to say that the spreads are "about equal", by comparing the heights of the boxes on the boxplots:

```
t.test(distance ~ gender, data = parking)
```

```
##
##  Welch Two Sample t-test
##
## data:  distance by gender
## t = -1.3238, df = 79.446, p-value =
## 0.1894
## alternative hypothesis: true difference in means is not equal to 0
## 95 percent confidence interval:
##  -4.5884103  0.9228228
## sample estimates:
## mean in group female   mean in group male
##             9.308511            11.141304
```

This is the Welch-Satterthwaite version of the test, the one that does not assume equal SDs in the two groups. The P-value of 0.1894 is not small, so there is no evidence of any difference in parking accuracy between males and females.

Or, this being the pooled one:

```
t.test(distance ~ gender, data = parking, var.equal = T)
```

```
##
##  Two Sample t-test
##
## data:  distance by gender
## t = -1.329, df = 91, p-value = 0.1872
## alternative hypothesis: true difference in means is not equal to 0
## 95 percent confidence interval:
##  -4.5722381  0.9066506
## sample estimates:
## mean in group female   mean in group male
##             9.308511            11.141304
```

You might have thought, looking at the boxplots, that the groups had about the same SD (based, for example, on noting that the two boxes were about the same height, so the IQRs were about the same). In that case, you might run a pooled t-test, which here gives an almost identical P-value of 0.1872, and the exact same conclusion.

(f) Why might you have some doubts about the t-test that you just did? Explain briefly.

Solution

The two-sample t-test is based on an assumption of normally-distributed data within each group. If you go back and look at the boxplots, you'll see either (depending on your point of view) that both groups are right-skewed, or that both groups have outliers, neither of which fits a normal distribution. The outlier in the male group is particularly egregious. So I think we are entitled to question whether a two-sample t-test is the right thing to do. Having said that, we should go back and remember that the t-tests are "robust to departures from normality" (since we are working with the Central Limit Theorem here), and therefore that this test might be quite good even though the data are not normal, because the sample sizes of 40-plus are large (by the standards of what typically makes the Central Limit Theorem work for us). So it may not be as bad as it seems. A common competitor for the two-sample t-test is the Mann-Whitney test. This doesn't assume normality, but it *does* assume symmetric distributions, which it's not clear that we have here. I like a test called Mood's Median Test, which is kind of the two-sample equivalent of the sign test (which we will also see later). It goes like this: Work out the overall median of all the distances, regardless of gender:

Google defines this as meaning *outstandingly bad, shocking*.

```
parking %>% summarize(med = median(distance))
```

```
## # A tibble: 1 x 1
##     med
##   <dbl>
## 1     9
```

The overall median is 9.

Count up how many distances of each gender were above or below the overall median. (Strictly, I'm supposed to throw away any values that are exactly equal to the overall median, but I won't here for clarity of exposition.)

```
tab = with(parking, table(gender, distance < 9))
tab
```

```
##
## gender   FALSE TRUE
##   female    23   24
##   male      27   19
```

For example, 19 of the male drivers had a distance (strictly) less than 9. Both genders are pretty close to 50–50 above and below the overall median, which suggests that the males and females have about the same median. This can be tested (it's a chi-squared test for independence, if you know that):

```
chisq.test(tab, correct = F)
```

```
##
##  Pearson's Chi-squared test
##
## data:  tab
## X-squared = 0.89075, df = 1, p-value =
## 0.3453
```

This is even less significant (P-value 0.3453) than the two-sample *t*-test, and so is consistent with our conclusion from before that there is actually no difference between males and females in terms of average parking distance. The Mood's median test is believable because it is not affected by outliers or distribution shape.

(g) The Toronto Star in its report said that females are more accurate at parking their cars. Why do you think they concluded that, and do you think they were right to do so? Explain briefly.

Solution

The conclusion from the boxplots was that the female median distance was less than the males, slightly, in this sample. That is probably what the Star seized on. Were they right? Well, that was why we did the test of significance. We were trying to see whether this observed difference between males and females was "real" (would hold up if you looked at "all" male and female drivers) or "reproducible" (you would expect to see it again if you did another study like this one). The large, non-significant P-values in all our tests tell us that the difference observed here was nothing more than chance. So it was not reasonable to conclude that females generally are more accurate at parallel-parking than males are.

6.3 *Bell peppers and too much water*

A pathogen called *Phytophthora capsici* causes bell peppers to wilt and die. It is thought that too much water aids in the spread of the

pathogen. Two fields are under study, labelled a and b. The first step in the research project is to compare the mean soil water content of the two fields. There is a suspicion that field a will have a higher water content than field b. The data are in the file link.

(a) Read the file in using read_csv, and list the resulting data frame.

Solution

Reading directly from the URL is easiest:

```
my_url = "http://www.utsc.utoronto.ca/~butler/d29/bellpepper.csv"
pepper = read_csv(my_url)
```

```
## Parsed with column specification:
## cols(
##   field = col_character(),
##   water = col_double()
## )
```

```
pepper
```

```
## # A tibble: 30 x 2
##    field water
##    <chr> <dbl>
##  1 a      10.2
##  2 a      10.7
##  3 a      15.5
##  4 a      10.4
##  5 a       9.9
##  6 a      10
##  7 a      16.6
##  8 a      15.1
##  9 a      15.2
## 10 a      13.8
## # ... with 20 more rows
```

If you like, find out how many observations you have from each field, thus:

```
pepper %>% count(field)
```

```
## # A tibble: 2 x 2
##   field     n
##   <chr> <int>
## 1 a        14
## 2 b        16
```

Fourteen and sixteen.

(b) Make side-by-side boxplots of the water content values for the two fields. How do the fields seem to compare?

Solution

This kind of thing:

```
ggplot(pepper, aes(x = field, y = water)) + geom_boxplot()
```

This one is rather interesting: the distribution of water contents for field `a` is generally higher than that for field `b`, but the median for `a` is actually *lower*.

The other reasonable plot is a facetted histogram, something like this:

```
ggplot(pepper, aes(x = water)) + geom_histogram(bins = 6) +
    facet_grid(field ~ .)
```

The distribution of water content in field `b` is actually bimodal, which is probably the explanation of the funny thing with the median. What actually seems to be happening (at least for these data) is

that the water content in field B is either about the same as field A, or a lot less (nothing in between). I can borrow an idea from earlier to find the five-number summaries for each field:

```
pepper %>% nest(-field) %>% mutate(qq = map(data,
    ~enframe(quantile(.$water)))) %>% unnest(qq) %>%
    mutate(pctile = parse_number(name)) %>% select(-name) %>%
    spread(pctile, value)
```

```
## # A tibble: 2 x 6
##   field   `0`  `25`  `50`  `75` `100`
##   <chr> <dbl> <dbl> <dbl> <dbl> <dbl>
## 1 a       9.9 10.5   11.4  14.8  16.6
## 2 b       7.1  8.33  11.8  12.6  13.9
```

This is a weird one: all the quantiles are greater for field A *except* for the median.

(c) Do a two-sample t-test to test whether there is evidence that the mean water content in field `a` is higher than that of field `b`. What do you conclude? Explain briefly. (You'll need to figure out a way of doing a one-sided test, or how to adapt the results from a two-sided test.)

Solution

```
t.test(water ~ field, alternative = "greater",
    data = pepper)
```

```
##
##  Welch Two Sample t-test
##
## data:  water by field
## t = 2.0059, df = 27.495, p-value =
## 0.0274
## alternative hypothesis: true difference in means is greater than 0
## 95 percent confidence interval:
##  0.2664399       Inf
## sample estimates:
## mean in group a mean in group b
##        12.52857        10.76875
```

Note the use of `alternative` to specify that the first group mean (that of field `a`) is bigger than the second, field `b`, under the alternative hypothesis.

The P-value, 0.0274, is less than 0.05, so we reject the null (equal means) in favour of the `a` mean being bigger than the `b` mean: field `a` really does have a higher mean water content.

Another way to tackle this is to do a two-sided test and adapt the P-value:

```
t.test(water ~ field, data = pepper)
```

```
##
##  Welch Two Sample t-test
##
## data:  water by field
## t = 2.0059, df = 27.495, p-value =
## 0.0548
## alternative hypothesis: true difference in means is not equal to 0
## 95 percent confidence interval:
##  -0.03878411  3.55842696
## sample estimates:
## mean in group a mean in group b
##        12.52857        10.76875
```

This time we do *not* go straight to the P-value. First we check that we are on the correct side, which we are since the sample mean for field `a` *is* bigger than for field `b`. Then we are entitled to take the two-sided P-value 0.0548 and *halve* it to get the same 0.0274 that we did before.

(d) Is the result of your test consistent with the boxplot, or not? Explain briefly.

Solution

The test said that field `a` had a greater mean water content. Looking at the boxplot, this is consistent with where the boxes sit (`a`'s box is higher up than `b`'s). However, it is not consistent with the medians, where `b`'s median is actually *bigger*. You have two possible right answers here: comparing the boxes with the test result (they agree) or comparing the medians with the test result (they disagree). Either is good. If you like, you could also take the angle that the two boxes overlap a fair bit, so it is surprising that the test came out significant. (The resolution of this one is that we have 30 measurements altogether, 14 and 16 in the two groups, so the sample size is not tiny. With smaller samples, having overlapping boxes would probably lead to a non-significant difference.)

6.4 *Exercise and anxiety and bullying mice*

Does exercise help to reduce anxiety? To assess this, some researchers randomly assigned mice to either an enriched environment where there was an exercise wheel available, or a standard environment

with no exercise options. After three weeks in the specified environment, for five minutes a day for two weeks, the mice were each exposed to a "mouse bully" — a mouse who was very strong, aggressive, and territorial. One measure of mouse anxiety is amount of time hiding in a dark compartment, with mice who are more anxious spending more time in darkness. The amount of time spent in darkness is recorded for each of the mice.

The data can be found at link.

(a) Read the data into R, and display your data frame. Count the number of mice in each group.

Solution

These are aligned columns with spaces in between, so we need `read_table`:

```
my_url = "http://www.utsc.utoronto.ca/~butler/d29/stressedmice.txt"
mice = read_table(my_url)
```

```
## Parsed with column specification:
## cols(
##   Time = col_integer(),
##   Environment = col_character()
## )
```

```
mice
```

```
## # A tibble: 14 x 2
##     Time Environment
##    <int> <chr>
##  1   359 Enriched
##  2   280 Enriched
##  3   138 Enriched
##  4   227 Enriched
##  5   203 Enriched
##  6   184 Enriched
##  7   231 Enriched
##  8   394 Standard
##  9   477 Standard
## 10   439 Standard
## 11   428 Standard
## 12   391 Standard
## 13   488 Standard
## 14   454 Standard
```

You can call the data frame whatever you like.

If you must, you can physically count the number of mice in each group, but you ought to get in the habit of coding this kind of thing:

```
mice %>% count(Environment)
```

```
## # A tibble: 2 x 2
##   Environment     n
##   <chr>       <int>
## 1 Enriched        7
## 2 Standard        7
```

Seven in each.

(b) Draw side-by-side boxplots of time spent in darkness for each group of mice.

Solution
This:

```
ggplot(mice, aes(x = Environment, y = Time)) +
    geom_boxplot()
```

You did remember to put capital letters on the variable names, didn't you?

(c) Do the boxplots support the hypothesis about exercise and anxiety? Explain briefly.

Solution
The hypothesis about exercise and anxiety is that mice who exercise more should be less anxious. How does that play out in this study? Well, mice in the enriched environment at least have the opportunity to exercise, which the mice in the standard environment do not, and anxiety is measured by the amount of time spent in darkness (more equals more anxious). So we'd expect the mice in the standard environment to spend more time in darkness, if that hypothesis is correct. That's exactly what the boxplots show, with very

little doubt. Your answer needs to make two points: (i) what you would expect to see, if the hypothesis about anxiety and exercise is true, and (ii) whether you actually did see it. You can do this either way around: for example, you can say what you see in the boxplot, and then make the case that this *does* support the idea of more exercise corresponding with less anxiety.

This means that I would expect to reject a null hypothesis of equal means, but I get ahead of myself.

(d) Carry out a t-test for comparing the mean time spent in darkness for the mice in the two groups. Think carefully about the details of the t-test (and what you need evidence in favour of).

Solution

We are trying to prove that exercise goes with *less* anxiety, so a one-sided test is called for. The other thing to think about is how R organizes the groups for `Environment`: in alphabetical order. Thus `Enriched` is first (like on the boxplot). We're trying to prove that the mean `Time` is *less* for `Enriched` than for `Standard`, so we need `alternative="less"`:

```
with(mice, t.test(Time ~ Environment, alternative = "less"))
```

```
##
##  Welch Two Sample t-test
##
## data:  Time by Environment
## t = -6.7966, df = 9.1146, p-value =
## 3.734e-05
## alternative hypothesis: true difference in means is less than 0
## 95 percent confidence interval:
##       -Inf -151.2498
## sample estimates:
## mean in group Enriched
##               231.7143
## mean in group Standard
##               438.7143
```

A common clue that you have the wrong alternative hypothesis is a P-value coming out close to *1*, which is what you would have gotten from something like this:

```
with(mice, t.test(Time ~ Environment, alternative = "greater"))
```

```
##
##  Welch Two Sample t-test
##
## data:  Time by Environment
```

```
## t = -6.7966, df = 9.1146, p-value = 1
## alternative hypothesis: true difference in means is greater than 0
## 95 percent confidence interval:
##  -262.7502       Inf
## sample estimates:
## mean in group Enriched
##               231.7143
## mean in group Standard
##               438.7143
```

Here, we looked at the pictures and expected to find a difference, so we expected to find a P-value close to 0 rather than close to 1.

(e) What do you conclude, in terms of anxiety and exercise (at least for mice)? Explain briefly.

Solution

The P-value (from the previous part) is 0.000037, which is way less than 0.05 (or 0.01 or whatever α you chose). So the null hypothesis (equal means) is resoundingly rejected in favour of the one-sided alternative that the mean anxiety (as measured by time spent in darkness) is less for the mice who (can) exercise. You need to end up by doing a one-sided test. An alternative to what I did is to do a two-sided test in the previous part. Then you can fix it up by recognizing that the means are the right way around for the research hypothesis (the mean time in darkness is way less for `Enriched`), and then dividing the two-sided P-value by 2. But you need to do the "correct side" thing: just halving the two-sided P-value is not enough, because the sample mean for `Enriched` might have been *more* than for `Standard`.

(f) Does anything in the previous parts suggest any problems with the analysis you just did? Explain briefly.

Solution

Look at the side-by-side boxplots. The strict assumptions hiding behind the t-tests are that the data in each group come from normal distributions (equal standard deviations are not required). Are the data symmetric? Are there any outliers? Well, I see a high outlier in the `Enriched` group, so I have some doubts about the normality. On the other hand, I only have seven observations in each group, so there is no guarantee even if the populations from which they come are normal that the samples will be. So maybe things are not so bad. This is one of those situations where you make a case and defend it. I don't mind so much which case you make, as long as you can defend it. Thus, something like either of these two is good:

- I see an outlier in the `Enriched` group. The data within each group are supposed to be normally distributed, and the `Enriched` group is not. So I see a problem.

- I see an outlier in the `Enriched` group. But the sample sizes are small, and an apparent outlier could arise by chance. So I do not see a problem.

Extra: another way to think about this is normal quantile plots to assess normality within each group. This uses the facetting trick to get a separate normal quantile plot for each `Environment`:

```
ggplot(mice, aes(sample = Time)) + stat_qq() +
    stat_qq_line() + facet_wrap(~Environment,
    scales = "free")
```

For the `Enriched` group, the upper-end outlier shows up. In a way this plot is no more illuminating than the boxplot, because you still have to make a call about whether this is "too big". Bear in mind also that these facetted normal quantile plots, with two groups, come out tall and skinny, so vertical deviations from the line are exaggerated. On this plot, the lowest value also looks too low.

For the `Standard` group, there are no problems with normality at all.

What happens if we change the shape of the plots?

```
ggplot(mice, aes(sample = Time)) + stat_qq() +
    stat_qq_line() + facet_wrap(~Environment,
    scales = "free", ncol = 1)
```

This makes the plots come out in one column, that is, short and squat. I prefer these. I'd still call the highest value in `Enriched` an outlier, but the lowest value now looks pretty close to what you'd expect.

6.5 *Diet and growth in boys*

A dietician is studying the effect of different diets on children's growth. In part of the study, the dietician is investigating two religious sects, labelled `a` and `b` in our data set. Both sects are vegetarian; the difference between them is that people in Sect A only eat vegetables from below the ground, and Sect B only eats vegetables from above the ground. The height and weight of the boys are measured at regular intervals. The data in link are the heights of the boys at age 12.

This was not sexism, but a recognition that boys and girls will be of different heights for reasons unrelated to diet. Doing it this way makes the analysis simpler.

(a) Read in the data and find out how many observations you have and which variables.

Solution

The data values are separated by one space, so:

```
my_url = "http://www.utsc.utoronto.ca/~butler/d29/kids-diet.txt"
diet = read_delim(my_url, " ")
```

```
## Parsed with column specification:
## cols(
##   sect = col_character(),
##   height = col_integer()
## )
```

```
diet
```

```
## # A tibble: 21 x 2
##    sect  height
##    <chr>  <int>
##  1 a        140
##  2 a        140
##  3 a        140
##  4 a        143
##  5 a        135
##  6 a        144
##  7 a        156
##  8 a        149
##  9 a        146
## 10 a        148
## # ... with 11 more rows
```

21 observations on two variables, `sect` and `height`. (You should state this; it is not enough to make the reader figure it out for themselves.)

The heights are evidently in centimetres.

You can call the data frame whatever you like.

(b) Obtain side-by-side boxplots of the heights for boys from each sect. Does it look as if the heights of the boys in each sect are different? Comment briefly.

Solution

The boxplot is the kind of thing we've seen before:

```
ggplot(diet, aes(x = sect, y = height)) + geom_boxplot()
```

It looks to me as if the boys in Sect B are taller on average.

(c) Looking at your boxplots, do you see any problems with doing a two-sample t-test? Explain briefly.

Solution

The assumption is that the data in each group are "approximately normal". Boxplots don't tell you about normality specifically, but they tell you whether there are any outliers (none here) and something about the shape (via the lengths of the whiskers). I'd say the Sect A values are as symmetric as we could hope for. For Sect B, you can say either that they're skewed to the left (and that therefore we have a problem), or that the heights are close enough to symmetric (and that therefore we don't). For me, either is good. As ever, normal quantile plots can offer more insight. With data in this form, the two samples are mixed up, but using facets is the way to go. Philosophically, we draw a normal quantile plot of *all* the heights, and then say at the end that we would actually like a separate plot for each sect:

```
diet %>% ggplot(aes(sample = height)) + stat_qq() +
    stat_qq_line() + facet_wrap(~sect, ncol = 1)
```

I decided that I wanted short squat plots rather than tall skinny ones.

With the sizes of the samples, I really don't see any problems here. Most of the evidence for the left skewness in Sect B is actually coming from that largest value being too small. Sect A is as good as you could wish for. Having extreme values being *not extreme enough* is not a problem, since it won't be distorting the mean.

The other way of doing this is to use `filter` to pull out the rows you want and then feed that into the plot:

```
secta = filter(diet, sect == "a") %>% ggplot(aes(sample = sect)) +
    stat_qq() + stat_qq_line()
```

and the same for sect B. This is the usual `ggplot`-in-pipeline thing where you don't have a named data frame in the `ggplot` because it will use whatever came out of the previous step of the pipeline.

(d) Run a *t*-test to determine whether the mean heights differ significantly. What do you conclude? Explain briefly. (Run the *t*-test even if your previous work suggests that it is not the right thing to do.)

Solution

The wording states that a two-sided test is correct, which is the default, so you don't need anything special:

```
t.test(height ~ sect, data = diet)
```

```
##
##  Welch Two Sample t-test
##
## data:  height by sect
## t = -1.7393, df = 14.629, p-value =
## 0.103
## alternative hypothesis: true difference in means is not equal to 0
## 95 percent confidence interval:
##  -12.007505   1.229728
## sample estimates:
## mean in group a mean in group b
##        144.8333        150.2222
```

This is a two-sample test, so it takes a `data=`.

Our null hypothesis is that the two sects have equal mean height. The P-value of 0.103 is larger than 0.05, so we do not reject that null hypothesis. That is, there is no evidence that the sects differ in mean height. (That is, our earlier thought that the boys in Sect B were taller is explainable by chance.)

You *must* end up with a statement about mean heights, and when you do a test, you must state the conclusion in the context of the problem, whether I ask you to or not. "Don't reject the null hypothesis" is a step on the way to an answer, not an answer in itself. If you think it's an answer in itself, you won't be of much use to the world as a statistician.

C32 folks might have been thinking that Mood's median test was the thing, if you were worried about that skewness in Sect B. My guess is that the *t*-test is the right, so it will be the better test (and give the smaller P-value) here, but if you want to do it, you could do it this way:

```
library(smmr)
median_test(diet, height, sect)
```

```
## $table
##      above
## group above below
##     a     4     7
##     b     6     3
##
## $test
##        what     value
## 1 statistic 1.8181818
## 2        df 1.0000000
## 3   P-value 0.1775299
```

My suspicion (that I wrote before doing the test) is correct: there is even less evidence of a difference in median height between the sects. The table shows that both sects are pretty close to 50–50 above and below the overall median, and with sample sizes this small, they are certainly not significantly different from an even split. The small frequencies bring a warning about the chi-squared approximation possibly not working (that `smmr` suppresses). We had one like this in C32, but there the result was very significant, and this one is very non-significant. However, the implication is the same: even if the P-value is not very accurate (because the expected frequencies for sect B are both 4.5), the conclusion is unlikely to be wrong because the P-value is so far from 0.05.

7
Power and sample size

```r
library(tidyverse)
```

```
## -- Attaching packages ---- tidyverse 1.2.1 --
```

```
## v ggplot2 3.1.0     v purrr   0.2.5
## v tibble  1.4.2     v dplyr   0.7.8
## v tidyr   0.8.1     v stringr 1.3.1
## v readr   1.1.1     v forcats 0.3.0
```

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

7.1 Simulating power

This question investigates power by simulation.

(a) Use `rnorm` to generate 10 random values from a normal distribution with mean 20 and SD 2. Do your values look reasonable? Explain briefly. (You don't need to draw a graph.)

Solution

`rnorm` with the number of values first, then the mean, then the SD:

```r
x = rnorm(10, 20, 2)
x
```

```
##  [1] 21.59476 18.64044 21.83231 18.76556
##  [5] 18.64861 21.81889 21.62614 20.18249
##  [9] 16.91266 20.63490
```

95% of the sampled values should be within 2 SDs of the mean, that is, between 16 and 24 (or 99.7% should be within 3 SDs of the mean, between 14 and 26). None of my values are even outside the interval 16 to 24, though yours may be different.

I saved mine in a variable and then displayed them, which you don't need to do. I did because there's another way of assessing them for reasonableness: turn the sample into z-scores and see whether the values you get look like z-scores (that is, most of them are between -2 and 2, for example):

```
(x - 20)/2
```

```
##  [1]  0.79738130 -0.67977910  0.91615386
##  [4] -0.61722168 -0.67569291  0.90944266
##  [7]  0.81307163  0.09124563 -1.54367207
## [10]  0.31744905
```

These ones look very much like z-scores. This, if you think about it, is really the flip-side of 68–95–99.7, so it's another way of implementing the same idea.

You might also think of finding the *sample* mean and SD, and demonstrating that they are close to the right answers. Mine are:

```
mean(x)
```

```
## [1] 20.06568
```

```
sd(x)
```

```
## [1] 1.731305
```

The sample SD is more variable than the sample mean, so it can get further away from the population SD than the sample mean does from the population mean.

The downside to this idea is that it doesn't get at assessing the normality, which looking at z-scores or equivalent does. Maybe coupling the above with a boxplot would have helped, had I not said "no graphs", since then you'd (hopefully) see no outliers and a roughly symmetric shape.

This is old-fashioned "base R" technology; you could do it with a data frame like this:

```
d = tibble(x = rnorm(10, 20, 2))
d
```

```
## # A tibble: 10 x 1
##        x
##    <dbl>
##  1  20.6
##  2  19.3
##  3  20.2
```

```
##  4  18.7
##  5  21.3
##  6  17.8
##  7  19.8
##  8  20.9
##  9  21.7
## 10  20.6
```

```
d %>% summarize(m = mean(x), s = sd(x))
```

```
## # A tibble: 1 x 2
##       m     s
##   <dbl> <dbl>
## 1  20.1  1.21
```

These are different random numbers, but are about equally what you'd expect. (These ones are a bit less variable than you'd expect, but with only ten values, don't expect perfection.)

Some discussion about the kind of values you should get, and whether or not you get them, is what is called for here. I want you to say something convincing about how the values you get come from a normal distribution with mean 20 *and* SD 2. "Close to 20" is not the whole answer here, because that doesn't get at "how close to 20?": that is, it talks about the mean but not about the SD.

(b) Estimate by simulation the power of a t-test to reject a null hypothesis of 20 when the true mean is also 20, the population SD is 2, and the sample size is 10, against a (default) two-sided alternative. Remember the steps: (i) generate a lot of random samples from the true distribution, (ii) run the t-test with the required null mean, (iii) pull out the P-values, (iv) count how many of them are 0.05 or less.

Solution

Once you get the hang of these, they all look almost the same. This one is easier than some because we don't have to do anything special to get a two-sided alternative hypothesis:

```
pvals <- rerun(1000, rnorm(10, 20, 2)) %>% map(~t.test(.,
    mu = 20)) %>% map_dbl("p.value")
tibble(pvals) %>% count(pvals <= 0.05)
```

```
## # A tibble: 2 x 2
##   `pvals <= 0.05`     n
##   <lgl>           <int>
## 1 FALSE             958
## 2 TRUE               42
```

The power is about 4.2%. This seems depressingly small, but see the next part.

(c) In the simulation you just did, was the null hypothesis true or false? Do you want to reject the null hypothesis or not? Explain briefly why the simulation results you got were (or were not) about what you would expect.

Solution

The null mean and the true mean were both 20: that is, the null hypothesis was correct, and rejecting it would be a mistake, to be precise a type I error. We were doing the test at $\alpha = 0.05$ (by comparing our collection of simulated P-values with 0.05), so we should be making a type I error 5% of the time. This is entirely in line with the 4.2% of (wrong) rejections that I had. Your estimation is likely to be different from mine, but you should be rejecting about 5% of the time. If your result is very different from 5%, that's an invitation to go back and check your code. On the other hand, if it *is* about 5%, that ought to give you confidence to go on and use the same ideas for the next part.

(d) By copying, pasting and editing your code from the previous part, estimate the power of the test of $H_0 : \mu = 20$ (against a two-sided alternative) when the true population mean is 22 (rather than 20).

Solution

Here's the code we just used:

```
pvals <- rerun(1000, rnorm(10, 20, 2)) %>% map(~t.test(.,
    mu = 20)) %>% map_dbl("p.value")
tibble(pvals) %>% count(pvals <= 0.05)
```

One of those 20s needs to become 22. Not the one in the `t.test`, since the hypotheses have not changed. So we need to change the 20 in the `rnorm` line to 22, since that's where we're generating data from the true distribution. The rest of it stays the same:

```
pvals <- rerun(1000, rnorm(10, 22, 2)) %>% map(~t.test(.,
    mu = 20)) %>% map_dbl("p.value")
tibble(pvals) %>% count(pvals <= 0.05)
```

```
## # A tibble: 2 x 2
##   `pvals <= 0.05`     n
##   <lgl>           <int>
## 1 FALSE             202
## 2 TRUE              798
```

This time, we *want* to reject, since the null hypothesis is false. So look at the `TRUE` count of 798: the power is about $798/1000 \simeq 80\%$. We are very likely to correctly reject a null of 20 when the mean is actually 22.

Another way to reason that the power should be fairly large is to think about what kind of sample you are likely to get from the true distribution: one with a mean around 22 and an SD around 2. Thus the t-statistic should be somewhere around this (we have a sample size of 10):

```
t_stat = (22 - 20)/(2/sqrt(10))
t_stat
```

```
## [1] 3.162278
```

and the two-sided P-value should be about

```
2 * (1 - pt(t_stat, 10 - 1))
```

```
## [1] 0.01150799
```

Of course, with your actual data, you will sometimes be less lucky than this (a sample mean nearer 20 or a larger sample SD), but sometimes you will be luckier. But the suggestion is that most of the time, the P-value will be pretty small and you will end up correctly rejecting.

The quantity `t_stat` above, 3.16, is known to some people as an "effect size", and summarizes how far apart the null and true means are, relative to the amount of variability present (in the sampling distribution of the sample mean). As effect sizes go, this one is pretty large.

(e) Use R to calculate this power exactly (without simulation). Compare the exact result with your simulation.

Solution

This is `power.t.test`. The quantity `delta` is the difference between true and null means:

```
power.t.test(n = 10, delta = 22 - 20, sd = 2,
    type = "one.sample", alternative = "two.sided")
```

```
##
##      One-sample t test power calculation
##
##               n = 10
##           delta = 2
```

```
##              sd = 2
##       sig.level = 0.05
##           power = 0.8030962
##     alternative = two.sided
```

This, 0.803, is very close to the 0.798 I got from my simulation. Which makes me think I did them both right. This is not a watertight proof, though: for example, I might have made a mistake and gotten lucky somewhere. But it does at least give me confidence. Extra: when you estimate power by simulation, what you are doing is rejecting or not with a certain probability (which is the same for all simulations). So the number of times you actually *do* reject has a binomial distribution with n equal to the number of simulated P-values you got (1000 in my case; you could do more) and a p that the simulation is trying to estimate. This is inference for a proportion, exactly what `prop.test` does.

Recall that `prop.test` has as input:

- a number of "successes" (rejections of the null in our case)
- the number of trials (simulated tests)
- the null-hypothesis value of `p` (optional if you only want a CI)
- (optional) a confidence level `conf.level`.

In part (b), we knew that the probability of (incorrectly) rejecting should have been 0.05 and we rejected 42 times out of 1000:

```r
prop.test(42, 1000, 0.05)
```

```
##
##  1-sample proportions test with
##  continuity correction
##
## data:  42 out of 1000, null probability 0.05
## X-squared = 1.1842, df = 1, p-value =
## 0.2765
## alternative hypothesis: true p is not equal to 0.05
## 95 percent confidence interval:
##  0.03079269 0.05685194
## sample estimates:
##     p
## 0.042
```

Looking at the P-value, we definitely fail to reject that the probability of (incorrectly) rejecting is the 0.05 that it should be. Ouch. That's

true, but unnecessarily confusing. Look at the confidence interval instead, 0.031 to 0.057. The right answer is 0.05, which is inside that interval, so good.

In part (c), we didn't know what the power was going to be (not until we calculated it with `power.t.test`, anyway), so we go straight for a confidence interval; the default 95% confidence level is fine. We (correctly) rejected 798 times out of 1000:

```
prop.test(798, 1000)
```

```
##
##  1-sample proportions test with
##  continuity correction
##
## data:  798 out of 1000, null probability 0.5
## X-squared = 354.02, df = 1, p-value <
## 2.2e-16
## alternative hypothesis: true p is not equal to 0.5
## 95 percent confidence interval:
##  0.7714759 0.8221976
## sample estimates:
##     p
## 0.798
```

I left out the 3rd input since we're not doing a test, and ignore the P-value that comes out. (The default null proportion is 0.5, which often makes sense, but not here.)

According to the confidence interval, the estimated power is between 0.771 and 0.822. This interval definitely includes what we now know is the right answer of 0.803.

This might be an accurate enough assessment of the power for you, but if not, you can do more simulations, say 10,000:

```
pvals <- rerun(10000, rnorm(10, 22, 2)) %>% map(~t.test(.,
    mu = 20)) %>% map_dbl("p.value")
tibble(pvals) %>% count(pvals <= 0.05)
```

```
## # A tibble: 2 x 2
##   `pvals <= 0.05`     n
##   <lgl>           <int>
## 1 FALSE            2004
## 2 TRUE             7996
```

I copied and pasted my code again, which means that I'm dangerously close to turning it into a function, but anyway.

The confidence interval for the power is then

```
prop.test(7996, 10000)
```

```
##
##  1-sample proportions test with
##  continuity correction
##
## data:  7996 out of 10000, null probability 0.5
## X-squared = 3589.2, df = 1, p-value <
## 2.2e-16
## alternative hypothesis: true p is not equal to 0.5
## 95 percent confidence interval:
##  0.7915892 0.8073793
## sample estimates:
##      p
## 0.7996
```

that is, from 0.792 to 0.807, which once again includes the right answer of 0.803. The first interval, based on 1,000 simulations, has length 0.051, while this interval has length 0.015. The first interval is more than three times as long as the second, which is about what you'd expect since the first one is based on 10 times fewer simulations, and thus ought to be a factor of $\sqrt{10} \simeq 3.16$ times longer.

This means that you can estimate power as accurately as you like by doing a large enough number of simulations. Provided, that is, that you are prepared to wait a possibly long time for it to finish working!

7.2 *Calculating power and sample size for estimating mean*

We are planning a study to estimate a population mean. The population standard deviation is believed to be 20, and the population distribution is believed to be approximately normal. We will be testing the null hypothesis that the population mean is 100. Suppose the population mean is actually 110, and we want to determine how likely we are to (correctly) reject the null hypothesis in this case, using a two-sided (but one-sample) test with $\alpha = 0.05$.

(a) We will take a sample of size $n = 30$. Calculate the power of this test.

Solution

`power.t.test`. Fill in: sample size `n`, difference in means `delta` ($10 = 110 - 100$), population SD `sd`, type of test `type` (`one.sample`) and kind of alternative hypothesis `alternative` (`two.sided`). Leave out `power` since that's what we want:

```r
power.t.test(n = 30, delta = 10, sd = 20, type = "one.sample",
    alternative = "two.sided")
```

```
##
##      One-sample t test power calculation
##
##               n = 30
##           delta = 10
##              sd = 20
##       sig.level = 0.05
##           power = 0.7539627
##     alternative = two.sided
```

I meant "calculate" exactly rather than "estimate" (by simulation). Though if you want to, you can do that as well, thus:

```r
pvals <- rerun(1000, rnorm(30, 110, 20)) %>% map(~t.test(.,
    mu = 100)) %>% map_dbl("p.value")
tibble(pvals) %>% count(pvals <= 0.05)
```

```
## # A tibble: 2 x 2
##   `pvals <= 0.05`     n
##   <lgl>           <int>
## 1 FALSE             257
## 2 TRUE              743
```

That came out alarmingly close to the exact answer.

(b) Find the sample size necessary to obtain a power of at least 0.80 under these conditions. What sample size do you need? Explain briefly how your answer is consistent with (a).

Solution

Again, the implication is "by calculation". This time, in `power.t.test`, put in 0.80 for `power` and leave out `n`. The order of things doesn't matter (since I have named everything that's going into `power.t.test`):

```r
power.t.test(delta = 10, power = 0.8, sd = 20,
    type = "one.sample", alternative = "two.sided")
```

```
##
##      One-sample t test power calculation
##
##               n = 33.3672
##           delta = 10
##              sd = 20
##       sig.level = 0.05
##           power = 0.8
##     alternative = two.sided
```

To get sample size for power at least 0.80, we have to round 33.36 *up* to the next whole number, ie. $n = 34$ is needed. (A sample of size 33 wouldn't quite have enough power.)

This answer is consistent with (a) because a sample size of 30 gave a power a bit less than 0.80, and so to increase the power by a little (0.75 to 0.80), we had to increase the sample size by a little (30 to 34).

Estimating sample sizes by simulation is tricky, because the sample size has to be input to the simulation. That means your only strategy is to try different sample sizes until you find one that gives the right power.

In this case, we know that a sample of size 30 doesn't give quite enough power, so we have to up the sample size a bit. How about we try 40? I copied and pasted my code from above and changed 30 to 40:

```
pvals <- rerun(1000, rnorm(40, 110, 20)) %>% map(~t.test(.,
    mu = 100)) %>% map_dbl("p.value")
tibble(pvals) %>% count(pvals <= 0.05)
```

```
## # A tibble: 2 x 2
##   `pvals <= 0.05`     n
##   <lgl>           <int>
## 1 FALSE             130
## 2 TRUE              870
```

Now the power is a bit too big, so we don't need a sample size quite as big as 40. So probably our next guess would be 35. But before we copy and paste again, we should be thinking about making a function of it first, with the sample size as input. Copy-paste once more and edit:

```
sim_power = function(n) {
    pvals <- rerun(1000, rnorm(30, 110, 20)) %>%
        map(~t.test(., mu = 100)) %>% map_dbl("p.value")
    tibble(pvals) %>% count(pvals <= 0.05)
}
```

In the grand scheme of things, we might want to have the null and true means, population SD and α be inputs to the function as well, so that we have a more general tool, but this will do for now.

Let's run it with a sample size of 35:

```
sim_power(35)
```

```
## # A tibble: 2 x 2
##   `pvals <= 0.05`     n
```

```
##   <lgl>           <int>
## 1 FALSE             238
## 2 TRUE              762
```

and I'm going to call that good. (Because there is randomness in the estimation of the power, don't expect to get *too* close to the right answer. This one came out a fair bit less than the right answer; the power for $n = 35$ should be a bit *more* than 0.80.)

Now that you have the software to do it, you can see that figuring out a sample size like this, at least roughly, won't take very long: each one of these simulations takes maybe seconds to run, and all you have to do is copy and paste the previous one, and edit it to contain the new sample size before running it again. You're making the computer work hard while you lazily sip your coffee, but there's no harm in that: programmer's brain cells are more valuable than computer CPU cycles, and you might as well save your brain cells for when you really need them.

7.3 *Simulating power for proportions*

In opinion surveys (and other places), we are testing for a proportion p (for example, the proportion of people agreeing with some statement). Often, we want to know whether the proportion is "really" greater than 0.5.

That would mean assessing whether an observed proportion could be greater than 0.5 just by chance, or whether it is bigger enough than 0.5 to reject chance as a plausible explanation.

That would entail testing a null $H_0 : p = 0.5$ against an alternative $H_a : p > 0.5$. This is usually done by calculating the test statistic

$$z = \frac{\hat{p} - 0.5}{\sqrt{0.25/n}},$$

where $\hat{p}$ is the observed proportion in the sample, and getting a P-value from the upper tail of a standard normal distribution. (The 0.25 is $p(1-p)$ where $p = 0.5$.) This is what `prop.test` does, as we investigate shortly.

(a) Use `rbinom` to generate a random value from a binomial distribution with $n = 100$ and $p = 0.6$. There are three inputs to `rbinom`: the first one should be the number 1, and the second and third are the n and p of the binomial distribution.

Solution

I am doing some preparatory work that you don't need to do:

```
set.seed(457299)
```

By setting the "seed" for the random number generator, I guarantee that I will get the same answers every time I run my code below

(and therefore I can talk about my answers without worrying that they will change). Up to you whether you do this. You can "seed" the random number generator with any number you like. A lot of people use `1`. Mahinda seems to like `123`. Mine is an old phone number.

And so to work:

```
rbinom(1, 100, 0.6)
```

```
## [1] 60
```

I got exactly 60% successes this time. You probably won't get exactly 60, but you should get somewhere close. (If you use my random number seed and use the random number generator exactly the same way I did, you should get the same values I did.)

For fun, you can see what happens if you change the 1:

```
rbinom(3, 100, 0.6)
```

```
## [1] 58 57 55
```

Three random binomials, that happened to come out just below 60. We're going to leave the first input as 1, though, and let `rerun` handle "lots of sampled values" later.

(b) Using the random binomial that you generated just above, use `prop.test` to test whether it could reasonably have come from a binomial population with $n = 100$ and $p = 0.5$, or whether p is actually bigger than 0.5. (Of course, you know it actually did not come from a population with $p = 0.5$.) `prop.test` has, for us, four inputs, thus:

- the observed number of successes
- the `n` of the binomial distribution
- the null-hypothesis `p` of the binomial distribution
- the alternative hypothesis, here "greater"

Solution

I got exactly 60 successes, so I do this:

```
prop.test(60, 100, 0.5, alternative = "greater")
```

```
##
##  1-sample proportions test with
##  continuity correction
##
```

```
## data:  60 out of 100, null probability 0.5
## X-squared = 3.61, df = 1, p-value =
## 0.02872
## alternative hypothesis: true p is greater than 0.5
## 95 percent confidence interval:
##  0.5127842 1.0000000
## sample estimates:
##   p
## 0.6
```

The P-value should at least be fairly small, since 60 is a bit bigger than 50. (Think about tossing a coin 100 times; would 60 heads make you doubt the coin's fairness? The above says it should.)

(c) Run `prop.test` again, just as you did before, but this time save the result, and extract the piece of it called `p.value`. Is that the P-value from your test?

Solution

Copying and pasting:

```
p_test = prop.test(60, 100, 0.5, alternative = "greater")
p_test$p.value
```

```
## [1] 0.02871656
```

Yep, the same.

(d) Use `rerun` to estimate the power of a test of $H_0 : p = 0.5$ against $H_a : p > 0.5$ when $n = 500$ and $p = 0.56$, using $\alpha = 0.05$. There are three steps:

- use `rerun` to generate random samples from binomial distributions with $n = 500$ and $p = 0.56$, repeated "many" times (something like 1000 or 10,000 is good)
- use `map` to run `prop.test` on each of those random samples
- use `map_dbl` to extract the P-value for each test and save the results (in a vector called, perhaps, `pvals`).

So I lied: the fourth and final step is to count how many of those P-values are 0.05 or less.

Solution

The previous parts, using `rbinom` and `prop.test`, were meant to provide you with the ingredients for this part. The first step is to use `rbinom`. The first input is 1 since we only want one random binomial each time (the `rerun` will handle the fact that you actually want lots

of them). The second step runs `prop.test`; the first input to that is each one of the numbers of successes from the first step. (This is an implied for-each, with each of the simulated binomials playing the role of "it", in turn.). The last part is to pull out all the P-values and make a table of them, just like the example in class.

```
pvals <- rerun(10000, rbinom(1, 500, 0.56)) %>%
    map(~prop.test(., 500, 0.5, alternative = "greater")) %>%
    map_dbl("p.value")
tibble(pvals) %>% count(pvals <= 0.05)
```

```
## # A tibble: 2 x 2
##   `pvals <= 0.05`     n
##   <lgl>           <int>
## 1 FALSE            1491
## 2 TRUE             8509
```

The estimated power is about 85%. That is, if p is actually 0.56 and we have a sample of size 500, we have a good chance of (correctly) rejecting that $p = 0.5$.

Extra: It turns out that SAS can work out this power by calculation (using `proc power`). SAS says our power is also about 85%, as our simulation said. I was actually pleased that my simulation came out so close to the right answer.

In contrast to `power.t.test`, SAS's `proc power` handles power analyses for a lot of things, including analysis of variance, correlation and (multiple) regression. What these have in common is some normal-based theory that allows you (under assumptions of sufficiently normal-shaped populations) to calculate the exact answer (that is, the distribution of the test statistic when the *alternative* hypothesis is true). The case we looked at is one of those because of the normal approximation to the binomial: once n gets big, particularly if p is somewhere near 0.5, the binomial is very well approximated by a normal with the right mean and SD.

The moral of this story is that when you have a decently large sample, $n = 500$ in this case, p doesn't have to get very far away from 0.5 before you can correctly reject 0.5. Bear in mind that sample sizes for estimating proportions need to be larger than those for estimating means, so $n = 500$ is large without being huge. The practical upshot is that if you design a survey and give it to 500 (or more) randomly chosen people, the proportion of people in favour doesn't have to be much above 50% for you to correctly infer that it *is* above 50%, most of the time.

8
The sign test and Mood's median test

```
library(tidyverse)
```

```
## -- Attaching packages ---- tidyverse 1.2.1 --
```

```
## v ggplot2 3.1.0     v purrr   0.2.5
## v tibble  1.4.2     v dplyr   0.7.8
## v tidyr   0.8.1     v stringr 1.3.1
## v readr   1.1.1     v forcats 0.3.0
```

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

```
library(smmr)
```

8.1 Running a maze

A researcher is trying to design a maze that can be run by rats in about 60 seconds. One particular maze was run by a sample of 21 rats, with the times shown in link.

(a) Read the data into R. What (if anything) are the data values delimited by?

Solution

Take a look at the data file first. There is only one column of data, so you can treat it as being delimited by anything you like: a space, or a comma (the file can also be treated as a `.csv`), etc.:

```
myurl = "http://www.utsc.utoronto.ca/~butler/c32/maze.txt"
times = read_delim(myurl, " ")
```

```
## Parsed with column specification:
## cols(
##   time = col_double()
## )
```

```
times
```

```
## # A tibble: 21 x 1
##     time
##    <dbl>
##  1  38.4
##  2  46.2
##  3  62.5
##  4  38
##  5  62.8
##  6  33.9
##  7  50.4
##  8  35
##  9  52.8
## 10  60.1
## # ... with 11 more rows
```

(b) Run a sign test, doing it yourself as we did in class: count the number of values above and below 60, take the *smaller* of those, and find the probability of a value of that or smaller still on a binomial distribution with $n = 21$ and $p = 0.5$ (we have 21 data points), doubling the answer because the test is two-sided.

Solution
Count how many values are above and below 60:

```
times %>% count(time > 60)
```

```
## # A tibble: 2 x 2
##   `time > 60`     n
##   <lgl>       <int>
## 1 FALSE          16
## 2 TRUE            5
```

5 above and 16 below. Then find out how likely it is that a binomial with $n = 21, p = 0.5$ would produce 5 or fewer successes:

```
p = sum(dbinom(0:5, 21, 0.5))
p
```

```
## [1] 0.01330185
```

or if you prefer count upwards from 16:

```
sum(dbinom(16:21, 21, 0.5))
```

```
## [1] 0.01330185
```

and double it to get a two-sided P-value:

```
2 * p
```

```
## [1] 0.0266037
```

We'll compare this with `smmr` in a moment.

(c) Install my package `smmr`, if you haven't already. To do this, you first need to install the package `devtools` (if you haven't already), by going to the console and typing

```
install.packages("devtools")
```

When that's all done, install `smmr` thus:

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

That all needs to be done only once. Then, each R Studio session where you want to use `smmr` needs this:

```
library(smmr)
```

As usual, only the `library` thing only needs to be done every time.

When you have `smmr` installed, use `sign_test` from that package to re-run your sign test. Do you get the same P-value?

Solution

The sign test function takes a data frame, an (unquoted) column name from that data frame of data to test the median of, and a null median (which defaults to 0 if you omit it):

```
library(smmr)
sign_test(times, time, 60)
```

```
## $above_below
## below above
##    16     5
##
## $p_values
##   alternative    p_value
## 1       lower 0.01330185
## 2       upper 0.99640131
## 3   two-sided 0.02660370
```

This shows you two things: a count of the values below and above the null median, and then the P-values according to the various alternative hypotheses you might have.

In our case, we see again the 16 maze-running times below 60 seconds and 5 above (one of which was a long way above, but we don't care about that here). We were testing whether the median was different from 60, so we look at the two-sided P-value of 0.0266, which is exactly what we had before.

If `sign_test` doesn't work for you (perhaps because it needs a function `enquo` that you don't have), there is an alternative function `sign_test0` that doesn't use it. It requires as input a *column* of values (extracted from the data frame) and a null median, thus:

```
with(times, sign_test0(time, 60))
```

```
## $above_below
## below above
##    16     5
##
## $p_values
##   alternative    p_value
## 1       lower 0.01330185
## 2       upper 0.99640131
## 3   two-sided 0.02660370
```

The output should be, and here is, identical.

(d) Package `smmr` also has a function `pval_sign`, which has the same input as `sign_test`, but with the null median *first*. Run it on your data and see what it gives.

Solution

Try it and see:

```
pval_sign(60, times, time)
```

```
## [1] 0.0266037
```

The two-sided P-value, and that is all. We'll be using this in a minute.

Alternatively, there is also this, which needs a null median and a *column* as input:

```
with(times, pval_sign0(60, time))
```

```
## [1] 0.0266037
```

(e) Obtain a 95% confidence interval for the median based on these data. Do this two ways. First, use the trial and error way from class (either the try-lots-of-values way or the bisection way; either is good). Second, use `ci_median` from `smmr`. The latter takes as input a data frame, a column name (unquoted) and optionally a `conf.level` that defaults to 0.95.

Solution

The reason for showing you `pval_sign` in the previous part is that this is a building block for the confidence interval. What we do is to try various null medians and find out which ones give P-values less than 0.05 (outside the interval) and which ones bigger (inside). We know that the value 60 is outside the 95% CI, and the sample median is close to 50 (which we expect to be inside), so sensible values to try for the upper end of the interval would be between 50 and 60:

```
pval_sign(58, times, time)
```

```
## [1] 0.0266037
```

```
pval_sign(55, times, time)
```

```
## [1] 0.6636238
```

So, 55 is inside the interval and 58 is outside. I could investigate further in similar fashion, but I thought I would try a whole bunch of null medians all at once. That goes like this:

```
meds = seq(55, 58, 0.25)
meds
```

```
##  [1] 55.00 55.25 55.50 55.75 56.00 56.25
##  [7] 56.50 56.75 57.00 57.25 57.50 57.75
## [13] 58.00
```

```
pvals = map_dbl(meds, pval_sign, times, time)
data.frame(meds, pvals)
```

```
##     meds      pvals
## 1  55.00 0.66362381
## 2  55.25 0.38331032
## 3  55.50 0.26317596
## 4  55.75 0.18924713
## 5  56.00 0.18924713
## 6  56.25 0.18924713
## 7  56.50 0.07835388
## 8  56.75 0.07835388
## 9  57.00 0.07835388
## 10 57.25 0.07835388
## 11 57.50 0.07835388
## 12 57.75 0.02660370
## 13 58.00 0.02660370
```

So values for the median all the way up to and including 57.5 are in the confidence interval.

What `map_dbl` does is to take a vector of values, here the ones in `meds` (55 through 58 in steps of 0.25), feed them into a function, here `pval_sign`, one by one and gather together the results. `pval_sign` has two other inputs, `times` and `time`, which are added to `map_dbl` at the end. (They are the same no matter what median we are testing.) So, putting the calculated P-values side by side with the null medians they belong to shows you which medians are inside the confidence interval and which are outside.

The function is called `map_dbl` because *my* function called `pval_sign` that is called repeatedly returns a single decimal number (a `dbl`). There is also, for example, `map_chr` for repeatedly calling a function that returns a single piece of text, and plain `map` that is used when the repeatedly-called function returns a data frame.

Since you don't know about `map_dbl`, I didn't want to confuse things more than necessary, but now that you *do* know what it does, you might be in a better position to understand this more Tidyverse-flavoured code, with the `map_dbl` inside a `mutate` and the data frame created as we go:

```
tibble(meds = seq(55, 58, 0.25)) %>% mutate(pvals = map_dbl(meds,
    pval_sign, times, time))
```

```
## # A tibble: 13 x 2
##     meds  pvals
##    <dbl>  <dbl>
##  1  55   0.664
##  2  55.2 0.383
##  3  55.5 0.263
##  4  55.8 0.189
##  5  56   0.189
##  6  56.2 0.189
##  7  56.5 0.0784
##  8  56.8 0.0784
##  9  57   0.0784
## 10  57.2 0.0784
## 11  57.5 0.0784
## 12  57.8 0.0266
## 13  58   0.0266
```

Now for the other end of the interval. I'm going to do this a different way: more efficient, but less transparent. The first thing I need is a pair of values for the median: one inside the interval and one outside. Let's try 40 and 50:

```
pval_sign(40, times, time)
```

```
## [1] 0.00719738
```

```
pval_sign(50, times, time)
```

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

OK, so 40 is outside and 50 is inside. So what do I guess for the next value to try? I could do something clever like assuming that the relationship between hypothesized median and P-value is *linear*, and then guessing where that line crosses 0.05. But I'm going to assume *nothing* about the relationship except that it goes uphill, and therefore crosses 0.05 somewhere. So my next guess is halfway between the two values I tried before:

```
pval_sign(45, times, time)
```

```
## [1] 0.07835388
```

So, 45 is inside the interval, and my (slightly) improved guess at the bottom end of the interval is that it's between 40 and 45. So next, I try halfway between *those*:

```
pval_sign(42.5, times, time)
```

```
## [1] 0.0266037
```

42.5 is outside, so the bottom end of the interval is between 42.5 and 45.

What we are doing is narrowing down where the interval's bottom end is. We started by knowing it to within 10, and now we know it to within 2.5. So if we keep going, we'll know it as accurately as we wish.

This is called a "bisection" method, because at each step, we're dividing our interval by 2.

There is one piece of decision-making at each step: if the P-value for the median you try is greater than 0.05, that becomes the top end of your interval (as when we tried 45); if it is less, it becomes the bottom end (when we tried 42.5).

This all begs to be automated into a loop. It's not a `for`-type loop, because we don't know how many times we'll be going around. It's a `while` loop: keep going while something is true. Here's how it goes:

```
lo = 40
hi = 50
while (abs(hi - lo) > 0.1) {
    try = (hi + lo)/2
    ptry = pval_sign(try, times, time)
```

```
    print(c(try, ptry))
    if (ptry < 0.05) {
        lo = try
    } else {
        hi = try
    }
}
```

```
## [1] 45.00000000  0.07835388
## [1] 42.5000000  0.0266037
## [1] 43.7500000  0.0266037
## [1] 44.37500000  0.07835388
## [1] 44.0625000  0.0266037
## [1] 44.2187500  0.0266037
## [1] 44.2968750  0.0266037
```

```
lo
```

```
## [1] 44.29688
```

```
pval_sign(lo, times, time)
```

```
## [1] 0.0266037
```

```
hi
```

```
## [1] 44.375
```

```
pval_sign(hi, times, time)
```

```
## [1] 0.07835388
```

The loop stopped because 44.297 and 44.375 are less than 0.1 apart. The first of those is outside the interval and the second is inside. So the bottom end of our interval is 44.375, to this accuracy. If you want it more accurately, change 0.1 in the `while` line to something smaller (but then you'll be waiting longer for the answer).

I put the `print` statement in the loop so that you could see what values were being tried, and what P-values they were producing. What happens with these is that the P-value jumps at each data value, so you won't get a P-value exactly 0.05; you'll get one above and one below.

Likewise, you can use the function with a zero on its name and feed it a column rather than a data frame and a column name:

```
meds = seq(55, 58, 0.25)
meds
```

```
##  [1] 55.00 55.25 55.50 55.75 56.00 56.25
##  [7] 56.50 56.75 57.00 57.25 57.50 57.75
## [13] 58.00
```

```
pvals = map_dbl(meds, ~with(times, pval_sign0(.,
    time)))
data.frame(meds, pvals)
```

```
##     meds      pvals
## 1  55.00 0.66362381
## 2  55.25 0.38331032
## 3  55.50 0.26317596
## 4  55.75 0.18924713
## 5  56.00 0.18924713
## 6  56.25 0.18924713
## 7  56.50 0.07835388
## 8  56.75 0.07835388
## 9  57.00 0.07835388
## 10 57.25 0.07835388
## 11 57.50 0.07835388
## 12 57.75 0.02660370
## 13 58.00 0.02660370
```

Or adapt the idea I had above for bisection. All that was a lot of work, but I wanted you to see it all once, so that you know where the confidence interval is coming from. `smmr` also has a function `ci_median` that does all of the above without you having to do it. As I first wrote it, it was using the trial and error thing with `map_dbl`, but I chose to rewrite it with the bisection idea, because I thought that would be more accurate.

```
ci_median(times, time)
```

```
## [1] 44.30747 57.59766
```

This is a more accurate interval than we got above. (The `while` loop for the bisection keeps going until the two guesses at the appropriate end of the interval are less than 0.01 apart, by default.)

You can change this by adding something like *tol=1e-4* to the end of your *ci-median*.

If you want some other confidence level, you add `conf.level` on the end, as you would for `t.test`:

```
ci_median(times, time, conf.level = 0.75)
```

```
## [1] 46.20444 55.49473
```

A 75% CI, just for fun. This is a shorter interval than the 95% one, as it should be.

Likewise there is a `ci_median0` that takes a column and an optional confidence level:

```
with(times, ci_median0(time))
```

```
## [1] 44.30747 57.59766
```

```
with(times, ci_median0(time, conf.level = 0.75))
```

```
## [1] 46.20444 55.49473
```

with the same results. Try `ci_median` first, and if it doesn't work, try `ci_median0`.

8.2 *Chocolate chips*

A famous cookie manufacturer claims that their bags of chocolate chip cookies contain "more than 1100 chocolate chips on average". A diligent group of students buys 16 bags of these cookies and counts the number of chocolate chips in each bag. The results are in `http://www.utsc.utoronto.ca/~butler/c32/chips.txt`.

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

Solution

I'll pretend it's a `.csv` this time, just for fun. Give the data frame a name different from `chips`, so that you don't get confused:

```
bags = read_csv("chips.txt")
```

```
## Parsed with column specification:
## cols(
##   chips = col_integer()
## )
```

```
bags
```

```
## # A tibble: 16 x 1
##    chips
##    <int>
##  1  1219
##  2  1214
##  3  1087
##  4  1200
##  5  1419
##  6  1121
##  7  1325
##  8  1345
##  9  1244
## 10  1258
```

```
## 11  1356
## 12  1132
## 13  1191
## 14  1270
## 15  1295
## 16  1135
```

That looks sensible.

(b) Build your own sign test in R for testing that the median is 1100 chocolate chips, against the alternative that it is greater. (Do this as in class: count the appropriate thing, compare it with an appropriate binomial distribution, and obtain a P-value.

Solution

The null median is 1100, so we count the number of values above and below:

```
bags %>% count(chips < 1100)
```

```
## # A tibble: 2 x 2
##   `chips < 1100`     n
##   <lgl>          <int>
## 1 FALSE             15
## 2 TRUE               1
```

The un-standard thing there is that we can put a logical condition directly into the `count`. If you don't think of that, you can also do this, which creates a new variable `less` that is `TRUE` or `FALSE` for each bag appropriately:

```
bags %>% mutate(less = (chips < 1100)) %>% count(less)
```

```
## # A tibble: 2 x 2
##   less      n
##   <lgl> <int>
## 1 FALSE    15
## 2 TRUE      1
```

or the more verbose

```
bags %>% mutate(less = (chips < 1100)) %>% group_by(less) %>%
    summarize(howmany = n())
```

```
## # A tibble: 2 x 2
##   less  howmany
##   <lgl>   <int>
## 1 FALSE      15
## 2 TRUE        1
```

Just one value below, with all the rest above. Getting the right P-value, properly, requires some careful thought (but you will probably get the right answer anyway). If the alternative hypothesis is true, and the median is actually bigger than 1100 (say, 1200), you would expect half the data values to be bigger than 1200 and half smaller. So *more* than half the data values would be bigger than *1100*, and fewer than half of them would be less than 1100. So, if we are going to reject the null (as it looks as if we will), that small number of values below 1100 is what we want.

The P-value is the probability of a value 1 or less in a binomial distribution with $n = 16, p = 0.5$:

```
sum(dbinom(0:1, 16, 0.5))
```

```
## [1] 0.0002593994
```

Or, equivalently, count *up* from 15:

```
sum(dbinom(15:16, 16, 0.5))
```

```
## [1] 0.0002593994
```

This is correctly one-sided, so we don't have to do anything with it.

(c) Use my R package `smmr` to reproduce your sign test above, and verify that you get consistent results. (See the maze-design question for instructions on installing this, if you haven't yet.)

Solution

This will mean reading the output carefully:

```
library(smmr)
sign_test(bags, chips, 1100)
```

```
## $above_below
## below above
##     1    15
##
## $p_values
##   alternative      p_value
## 1       lower 0.9999847412
## 2       upper 0.0002593994
## 3   two-sided 0.0005187988
```

This time, we're doing a one-sided test, specifically an *upper-tail* test, since we are looking for evidence that the median is *greater than* 1100. The results are exactly what we got "by hand": 15 values above

and one below, and a P-value (look along the `upper` line) of 0.00026. The two-sided P-value of 0.00052 rounds to the same 0.0005 as SAS got.

Alternatively, you can do this:

```
sign_test0(bags$chips, 1100)
```

```
## $above_below
## below above
##     1    15
##
## $p_values
##   alternative      p_value
## 1       lower 0.9999847412
## 2       upper 0.0002593994
## 3   two-sided 0.0005187988
```

with the same result (but only go this way if you need to).

(d) Use `smmr` to obtain a 95% confidence interval for the median number of chocolate chips per bag of cookies.

Solution

Once everything is in place, this is simplicity itself:

```
ci_median(bags, chips)
```

```
## [1] 1135.003 1324.996
```

1135 to 1325. I would round these off to whole numbers, since the data values are all whole numbers. These values are all above 1100, which supports the conclusion we got above that the median is above 1100. This is as it should be, because the CI is "all those medians that would *not* be rejected by the sign test".

Or,

```
ci_median0(bags$chips)
```

```
## [1] 1135.003 1324.996
```

8.3 *The power of the sign test*

I've mentioned several times that the sign test has less power than the t-test. Let's investigate this with a specific example.

Let's suppose we are testing $H_0 : \mu = 40$ against $H_a : \mu \neq 40$, where μ is the population mean (and median, as we shall see). Our population actually has a normal distribution with mean 50 and SD

15, so that the null hypothesis is *wrong* and we want to reject it most of the time. On the other hand, the population actually *is* normally-distributed and so the t-test is the right one to use.

(This is an old question, so I tackle the simulated power differently than I did it in class this time. But see if you can follow what I do here.)

(a) Use `power.t.test` to find the probability that a t-test correctly rejects the null hypothesis using a sample size of $n = 10$.

Solution

```
power.t.test(delta = 50 - 40, n = 10, sd = 15,
    type = "one.sample", alternative = "two.sided")
```

```
##
##      One-sample t test power calculation
##
##               n = 10
##           delta = 10
##              sd = 15
##       sig.level = 0.05
##           power = 0.4691805
##     alternative = two.sided
```

The power is 0.469. Not great, but we'll see how this stacks up against the sign test.

(b) What code in R would draw a random sample of size 10 from the *true* population distribution and save the sample in a variable?

Solution

The data actually have a normal distribution with mean 50 and SD 15, so we use `rnorm` with this mean and SD, obtaining 10 values:

```
x = rnorm(10, 50, 15)
x
```

```
##  [1] 57.00173 61.62367 77.65209 44.47180
##  [5] 49.09039 28.85521 32.01528 61.65065
##  [9] 65.96417 70.09647
```

(c) What code would count how many of the sampled values are less than 40 and how many are greater (or equal)?

Solution

The way we know this is to put `x` into a data frame first:

```
tibble(x) %>% count(x < 40)
```

```
## # A tibble: 2 x 2
##   `x < 40`     n
##   <lgl>    <int>
## 1 FALSE        8
## 2 TRUE         2
```

2 values less (and 8 greater-or-equal).

(d) It turns out the sign test would reject $H_0 : M = 40$ against $H_a : M \neq 40$ at $\alpha = 0.05$ if the smaller of the numbers in the last part is 1 or less. (M is the population median.) Add to your pipeline to obtain `TRUE` if you should reject the null for your data and `FALSE` otherwise.

Solution

This is actually easier than you might think. The output from `count` is a data frame with a column called `n`, whose minimum value you want. I add to my pipeline:

```
tibble(x) %>% count(x < 40) %>% summarize(the_min = min(n)) %>%
    mutate(is_rejected = (the_min <= 1))
```

```
## # A tibble: 1 x 2
##   the_min is_rejected
##     <dbl> <lgl>
## 1       2 FALSE
```

This will fail sometimes. If all 10 of your sample values are greater than 40, which they might turn out to be, you'll get a table with only one line, `FALSE` and 10; the minimum of the `n` values is 10 (since there is only one), and it will falsely say that you should not reject. The fix is

```
tibble(x) %>% count(x < 40) %>% summarize(the_min = min(n)) %>%
    mutate(is_rejected = (the_min <= 1 | the_min ==
        10))
```

```
## # A tibble: 1 x 2
##   the_min is_rejected
##     <dbl> <lgl>
## 1       2 FALSE
```

The above is almost the right thing, but not quite: we only want that value that I called `is_rejected`, rather than the whole data frame, so a `pull` will grab it:

```
tibble(x) %>% count(x < 40) %>% summarize(the_min = min(n)) %>%
    mutate(is_rejected = (the_min <= 1 | the_min ==
        10)) %>% pull(is_rejected)
```

```
## [1] FALSE
```

You might be wondering where the "1 or less" came from. Getting a P-value for the sign test involves the binomial distribution: if the null is correct, each data value is independently either above or below 40, with probability 0.5 of each, so the number of values below 40 (say) is binomial with $n = 10$ and $p = 0.5$. The P-value for 1 observed value below 40 and the rest above is

```
2 * pbinom(1, 10, 0.5)
```

```
## [1] 0.02148438
```

which is less than 0.05; the P-value for 2 values below 40 and the rest above is

```
2 * pbinom(2, 10, 0.5)
```

```
## [1] 0.109375
```

which is bigger than 0.05.

You might have encountered the term "critical region" for a test. This is the values of your test statistic that you would reject the null hypothesis for. In this case, the critical region is 1 and 0 observations below 40, along with 1 and 0 observations above 40.

When you're thinking about power, I think it's easiest to think in terms of the critical region (rather than directly in terms of P-values) since you have a certain α in mind all the way through, 0.05 in the power examples that I've done. The steps are then:

- Work out the critical region for your test, the values of the test statistic (or sample mean or sample count) that would lead to rejecting the null hypothesis.
- Under your particular alternative hypothesis, find the probability of falling into your critical region.

When I say "work out", I mean either calculating (along the lines of STAB57), or simulating, as we have done here.

(e) Use `rerun` to simulate the above process 1000 times: drawing a random sample from a normal distribution with mean 50 and SD 15, counting the number of values below and above 40, rejecting if the minimum of those is 1 or less, then counting the number of rejections out of 1000.

Solution

The way we've used `rerun` is to use it to select the random samples, and then we use `map` ideas to do what we want to do with each random sample, along the lines of what we did with the one random sample above. This is liable to go wrong the first few times, so make sure that each line works before you go on to the next. (`rerun` will produce you a `list` of random samples, with each of which you want to do something.) While you're debugging, try it with a small number of random samples like 5. I start with setting the random number seed, so it comes out the same each time. I discuss the results below and the code below that.

```
set.seed(457299)
rerun(1000, rnorm(10, 50, 15)) %>% map(~tibble(x = .)) %>%
    map(~count(., x < 40)) %>% map(~summarize(.,
    the_min = min(n))) %>% map(~mutate(., is_rejected = (the_min <=
    1 | the_min == 10))) %>% map_lgl(~pull(.,
    is_rejected)) %>% tibble(was_true = .) %>%
    count(was_true)
```

```
## # A tibble: 2 x 2
##   was_true     n
##   <lgl>    <int>
## 1 FALSE      757
## 2 TRUE       243
```

The estimated power of the sign test is 0.243, since that was the number of times a simulated sample gave us 0 or 1 values above or below 40 (and the rest on the other side).

All right, that code is seriously scary. Let me try to take you through it.

- The `rerun` line is the same kind of thing we had before: generate 1000 different random samples of size 10 from a normal distribution with mean 50 and SD 15.
- The output from the previous step is a `list` of vectors. But we like data frames to count things in, so for each vector we turn it into a data frame, filling a column called `x` with whatever was in each vector (that random sample). So now we have 1000 data frames each containing a column called `x`.
- Next, in each of those data frames, count up how many of the `x` values are less than 40. This will produce a data frame each time containing a column `n` that is the frequencies. Here and below, the `.` is used to denote "it": that is, each of the elements of the

list created originally by `rerun` that we are doing something with. Also, at the moment, the output for each element of the list is a data frame, so we stick with `map` for the time being.

- Next, for each of those tables of frequencies, find the smallest one and call it `the_min`. (As discussed above, all the values might be bigger than 40, in which case `the_min` will be 10 rather than 0, which we handle next.)
- Next, we create a new column called `is_rejected` which says that we should reject a median of 40 if the minimum value we just calculated is 1 or less, or if it happens to be 10, in which case that would have been the only entry in the frequency table, so that the missing one would have been zero.
- Next, we pull out only the true-or-false value in `is_rejected`. At last, the answer here is not a data frame but a simple logical value; `map_lgl` is like `map_dbl` except that the thing we are doing returns a `TRUE` or a `FALSE` rather than a number.
- At this point we have a vector of 1000 true-or-false. We want to count them, so we put them into a data frame (with a column called `was_true`), and in the last line, count them up. There are 243 (correct) rejections and 757 (incorrect) non-rejections.

You may now breathe again.

I'm now thinking a better way to do this is to write a function that takes a sample (in a vector) and returns a TRUE or FALSE according to whether or not a median of 40 would be rejected for that sample:

```
is_reject = function(x) {
    tibble(x = x) %>% count(x < 40) %>% summarize(the_min = min(n)) %>%
        mutate(is_rejected = the_min <= 1 | the_min ==
            10) %>% pull(is_rejected)
}
```

Now, we have to use that. This function will be "mapped" over for each of the random samples that come out of `rerun`, but now there will be only one `map` because the complication of the multiple `maps` has been subsumed into this one function. I'll set my random number seed so that I get the same results as before:

```
set.seed(457299)
rerun(1000, rnorm(10, 50, 15)) %>% map_lgl(~is_reject(.)) %>%
    tibble(rejected = .) %>% count(rejected)
```

```
## # A tibble: 2 x 2
##   rejected     n
```

```
##   <lgl>    <int>
## 1 FALSE      757
## 2 TRUE       243
```

Same results, and yeah, I like that a lot better.

(f) Which is more powerful in this case, the sign test or the t-test? How do you know?

Solution

The power of the sign test is estimated as 0.243, which is quite a bit less than the power of the t-test, which we found back in (a) to be 0.469. So the t-test, in this situation where it is valid, is the right test to use: it is (i) valid and (ii) more powerful. So the t-test is more powerful. One way to think about how *much* more powerful is to ask "how much smaller of a sample size would be needed for the t-test to have the same power as this sign test?" The power of my sign test was 0.243, so in `power.t.test` we set `power` equal to that and omit the sample size `n`:

```r
power.t.test(delta = 50 - 40, power = 0.243, sd = 15,
    type = "one.sample", alternative = "two.sided")
```

```
##
##      One-sample t test power calculation
##
##               n = 5.599293
##           delta = 10
##              sd = 15
##       sig.level = 0.05
##           power = 0.243
##     alternative = two.sided
```

A sample of size 6 gives the same power for the t-test that a sample of size 10 does for the sign test. The ratio of these two sample sizes is called the *relative efficiency* of the two tests: in this case, the t-test is $10/6 = 1.67$ times more efficient. The data that you have are being used "more efficiently" by the t-test. It is possible to derive

Meaning, I forget how to do it. But it has something to do with looking at alternatives that are very close to the null.

the limiting relative efficiency of the t test relative to the sign test when the data are actually normal, as the sample size gets larger. This turns out not to depend on how far wrong the null is (as long as it is the same for both the t-test and the sign test). This "asymptotic relative efficiency" is $\pi/2 = 1.57$. Our relative efficiency for power 0.243, namely 1.67, was pretty close to this, even though our sample sizes 10 and 6 are not especially close to infinity. This says that, if your data are actually from a normal distribution, you do a lot better

to use the t-test than the sign test, because the sign test is wasteful of data (it only uses above/below rather than the actual values). If your data are *not* from a normal distribution, then the story can be very different. Of course you knew I would investigate this. There is a distribution called the "Laplace" or "double exponential" distribution, that has very long tails. The distribution is not in base R, but there is a package called `smoothmest` that contains a function `rdoublex` to generate random values from this distribution. So we're going to do a simulation investigation of the power of the sign test for Laplace data, by the same simulation technique that we did above. Like the normal, the Laplace distribution is symmetric, so its mean and median are the same (which makes our life easier).

If you've ever run into the exponential distribution, you'll recall that this is right skewed with a very long tail. The Laplace distribution looks like two of these glued back to back.

Let's test the hypothesis that the median is zero. We'll suppose that the true median is 0.5 (this is called `mu` in `rdoublex`). The first problem we run into is that we can't use `power.t.test` because they assume normal data, which we are far from having. So we have to do two simulations: one to simulate the power of the t test, and one to simulate the power of the sign test.

This is about the *only* way in which the normal and Laplace distributions are alike.

To simulate the t test, we first have to generate some Laplace data with the true mean of 0.5. We'll use a sample size of 50 throughout these simulations.

```
library(smoothmest)

## Loading required package: MASS

##
## Attaching package: 'MASS'

## The following object is masked from 'package:dplyr':
##
##     select

rl = rdoublex(50, mu = 0.5)
rl

##  [1] -0.33323285  0.70569291 -1.22513053
##  [4]  0.68517708  0.12778518  0.50749949
##  [7]  0.26700527  1.90236874  0.53288312
## [10] -0.37374732  0.27256566  0.53365929
## [13]  0.43581431 -0.01545866  0.18594908
## [16] -0.40403202  1.13540289  0.16137306
## [19] -0.23360644 -0.74050354  2.92089551
## [22] -2.72173880  0.48428815  1.23636045
## [25]  0.17078618  1.72456334  0.07903058
## [28]  0.25210411  0.09512810  2.52310082
```

```
## [31] -2.13629814  0.81851434  0.74615575
## [34] -0.26068744  2.70683355  1.46981530
## [37]  1.45646489 -0.20232517  6.65249860
## [40]  1.51575026 -0.07606399 -1.11338640
## [43] -1.20427995 -0.70986104 -1.66466321
## [46]  0.55346854  0.66091469  0.72100677
## [49]  0.92025176  0.98922656
```

This seems to have some unusual values, far away from zero:

```
tibble(rl) %>% ggplot(aes(sample = rl)) + stat_qq() +
    stat_qq_line()
```

You see the long tails compared to the normal.

Now, we feed these values into `t.test` and see whether we reject a null median of zero (at $\alpha = 0.05$):

```
tt = t.test(rl)
tt
```

```
##
##  One Sample t-test
##
## data:  rl
## t = 2.2556, df = 49, p-value = 0.0286
## alternative hypothesis: true mean is not equal to 0
## 95 percent confidence interval:
##  0.04959344 0.85981931
## sample estimates:
## mean of x
## 0.4547064
```

Or we can just pull out the P-value and even compare it to 0.05:

```
pval = tt$p.value
pval
```

```
## [1] 0.02859596
```

```
is.reject = (pval <= 0.05)
is.reject
```

```
## [1] TRUE
```

$

This one has a small P-value and so the null median of 0 should be (correctly) rejected.

We'll use these ideas to simulate the power of the *t*-test for these data, testing a mean of 0. This uses the same ideas as for any power simulation; the difference here is the true distribution:

```
pvals <- rerun(1000, rdoublex(50, mu = 0.5)) %>%
    map(~t.test(., mu = 0)) %>% map_dbl("p.value")
```

and then count them:

```
tibble(pvals) %>% count(pvals <= 0.05)
```

```
## # A tibble: 2 x 2
##   `pvals <= 0.05`     n
##   <lgl>           <int>
## 1 FALSE             304
## 2 TRUE              696
```

And now we simulate the sign test. Since what we want is a P-value from a vector, the easiest way to do this is to use `pval_sign0` from `smmr`, which returns exactly the two-sided P-value that we want, so that the procedure is a step simpler:

```
pvals_sign <- rerun(1000, rdoublex(50, mu = 0.5)) %>%
    map_dbl(~pval_sign0(0, .))
```

and then

```
tibble(pvals_sign) %>% count(pvals_sign <= 0.05)
```

```
## # A tibble: 2 x 2
##   `pvals_sign <= 0.05`     n
##   <lgl>                <int>
## 1 FALSE                  239
## 2 TRUE                   761
```

For data from this Laplace distribution, the power of this t-test is 0.696, but the power of the sign test on the same data is 0.761, *bigger*. For Laplace-distributed data, the sign test is *more* powerful than the t-test.

This is not to say that you will ever run into data that comes from the Laplace distribution. But the moral of the story is that the sign test *can* be more powerful than the t-test, under the right circumstances (and the above simulation is the "proof" of that statement). So a blanket statement like "the sign test is not very powerful" needs to be qualified a bit: when your data come from a sufficiently long-tailed distribution, the sign test can be more powerful relative to the t-test than you would think.

I finish by "unloading" the two packages that got loaded:

```
detach(package:smoothmest, unload = T)
detach(package:MASS, unload = T)
```

8.4 *Sugar in breakfast cereals*

The data for this question are in `http://www.utsc.utoronto.ca/~butler/c32/cereal-sugar.txt`. The story here is whether breakfast cereals marketed to children have a lot of sugar in them; in particular, whether they have more sugar on average than cereals marketed to adults.

(a) Read in the data (to R) and display the data set. Do you have a variable that distinguishes the children's cereals from the adults' cereals, and another that contains the amount of sugar?

Solution

```
my_url = "http://www.utsc.utoronto.ca/~butler/c32/cereal-sugar.txt"
cereals = read_delim(my_url, " ")
```

```
## Parsed with column specification:
## cols(
##   who = col_character(),
##   sugar = col_double()
## )
```

```
cereals
```

```
## # A tibble: 40 x 2
##    who      sugar
##    <chr>    <dbl>
##  1 children  40.3
```

```
##  2 children  55
##  3 children  45.7
##  4 children  43.3
##  5 children  50.3
##  6 children  45.9
##  7 children  53.5
##  8 children  43
##  9 children  44.2
## 10 children  44
## # ... with 30 more rows
```

The variable `who` is a categorical variable saying who the cereal is intended for, and the variable `sugar` says how much sugar each cereal has.

(b) Calculate the mean sugar content for each group of cereals (the adults' ones and the children's ones). Do they look similar or different?

Solution
`group_by` and `summarize`:

```
cereals %>% group_by(who) %>% summarize(sugar_mean = mean(sugar))
```

```
## # A tibble: 2 x 2
##   who      sugar_mean
##   <chr>         <dbl>
## 1 adults         10.9
## 2 children       46.6
```

These means look very different, though it would be better to look at a boxplot (coming up in a moment).

(c) Make side-by-side boxplots of the sugar contents of the two types of cereal. What do you see that is out of the ordinary?

Solution
The usual:

```
ggplot(cereals, aes(x = who, y = sugar)) + geom_boxplot()
```

I see outliers: two high ones on the adults' cereals, and one high and one low on the children's cereals.

My thought above about the means being very different is definitely supported by the medians being very different on the boxplots. We should have no trouble declaring that the "typical" amounts of sugar in the adults' and children's cereals are different.

(d) Explain briefly why you would not trust a two-sample t-test with these data. (That is, say what the problem is, and why it's a problem.)

Solution

The problem is the outliers (which is rather a giveaway), but the reason it's a problem is that the two-sample t-test assumes (approximately) normal data, and a normal distribution doesn't have outliers. Not only do you need to note the outliers, but you also need to say why the outliers cause a problem *in this case*. Anything less than that is not a complete answer.

(e) Run a suitable test to see whether the "typical" amount of sugar differs between adult's and children's cereals. Justify the test that you run. (You can use the version of your test that lives in a package, if that is easier for you.) What do you conclude, in the context of the data?

Solution

Having ruled out the two-sample t-test, we are left with Mood's median test. I didn't need you to build it yourself, so you can use package `smmr` to run it with:

```
library(smmr)
median_test(cereals, sugar, who)
```

```
## $table
##           above
## group      above below
##   adults       2    19
##   children    18     1
##
## $test
##        what        value
## 1 statistic 2.897243e+01
## 2        df 1.000000e+00
## 3   P-value 7.341573e-08
```

We conclude that there *is* a difference between the median amounts of sugar between the two groups of cereals, the P-value of 0.00000007 being extremely small.

Why did it come out so small? Because the amount of sugar was smaller than the overall median for almost all the adult cereals, and larger than the overall median for almost all the children's ones. That is, the children's cereals really do have more sugar.

Mood's median test doesn't come with a confidence interval (for the difference in population medians), because whether or not a certain difference in medians is rejected depends on what those medians actually are, and the idea of the duality of the test and CI doesn't carry over as we would like.

My daughter likes chocolate Cheerios, but she also likes Shredded Wheat and *Bran Flakes*. Go figure. (Her current favourite is Raisin Bran, even though she doesn't like raisins by themselves.)

Mood's median test is the test we should trust, but you might be curious about how the *t*-test stacks up here:

```r
t.test(sugar ~ who, data = cereals)
```

```
##
##  Welch Two Sample t-test
##
## data:  sugar by who
## t = -11.002, df = 37.968, p-value =
## 2.278e-13
## alternative hypothesis: true difference in means is not equal to 0
## 95 percent confidence interval:
##  -42.28180 -29.13925
## sample estimates:
##   mean in group adults
##               10.90000
## mean in group children
##               46.61053
```

The P-value is *even smaller*, and we have the advantage of getting a confidence interval for the difference in means: from about 30 to about 40 units less sugar in the adult cereals. Whatever the units were.

8.5 Fear of math

Two new short courses have been proposed for helping students who suffer from severe math phobia. The courses are labelled A and B. Ten students were randomly allocated to one of these two courses, and each student's score on a math phobia test was recorded after they completed their course. The math phobia test produces whole-number scores between 0 and 10, with a higher score indicating a greater fear of mathematics. The data can be found in link. We start with R for this question.

(a) Read in the data and check, however you like, that you have 10 observations, 5 from each course.

Solution
This doesn't need much comment:

```
my_url = "http://www.utsc.utoronto.ca/~butler/c32/mathphobia.txt"
math = read_delim(my_url, " ")
```

```
## Parsed with column specification:
## cols(
##   course = col_character(),
##   phobia = col_integer()
## )
```

```
math
```

```
## # A tibble: 10 x 2
##    course phobia
##    <chr>   <int>
##  1 a           8
##  2 a           7
##  3 a           7
##  4 a           6
##  5 a           6
##  6 b           9
##  7 b           8
##  8 b           7
##  9 b           2
## 10 b           1
```

This will do, counting the a and b. Or, to save yourself that trouble:

```
math %>% count(course)
```

```
## # A tibble: 2 x 2
##   course     n
##   <chr>  <int>
## 1 a          5
## 2 b          5
```

Five each. The story is to get the computer to do the grunt work for you, if you can make it do so. Other ways:

```
math %>% group_by(course) %>% summarize(count = n())
```

```
## # A tibble: 2 x 2
##   course count
##   <chr>  <int>
## 1 a          5
## 2 b          5
```

and this:

```
with(math, table(course))
```

```
## course
## a b
## 5 5
```

giving the same answer. Lots of ways.

Extra: there is an experimental design issue here. You might have noticed that each student did only *one* of the courses. Couldn't students do both, in a matched-pairs kind of way? Well, it's a bit like the kids learning to read in that if the first of the courses reduces a student's anxiety, the second course won't appear to do much good (even if it actually would have been helpful had the student done that one first). This is the same idea as the kids learning to read: once you've learned to read, you've learned to read, and learning to read a second way won't help much. The place where matched pairs scores is when you can "wipe out" the effect of one treatment before a subject gets the other one. We have an example of kids throwing baseballs and softballs that is like that: if you throw one kind of ball, that won't affect how far you can throw the other kind.

(b) Do a two-sample t-test to assess whether there is a difference in mean phobia scores after the students have taken the two courses. What do you conclude? (You have no *a priori* reason to suppose that a particular one of the tests will produce a higher mean than the other, so do a two-sided test.)

That is, before looking at the data. This is Latin. It's also the place that the Bayesian *prior distribution* comes from. The *posterior distribution* comes from the Latin *a posteriori*, which means *afterwards*, that is, after you have looked at the data.

Solution

A two-sided test is the default, so there is not much to do here:

```
t.test(phobia ~ course, data = math)
```

```
##
##  Welch Two Sample t-test
##
## data:  phobia by course
## t = 0.83666, df = 4.4199, p-value =
## 0.4456
## alternative hypothesis: true difference in means is not equal to 0
## 95 percent confidence interval:
##  -3.076889  5.876889
## sample estimates:
## mean in group a mean in group b
##             6.8             5.4
```

The P-value of 0.4456 is nowhere near less than 0.05, so there is no evidence at all that the

mean math phobia scores are different between the two courses.

(c) Draw boxplots of the math phobia scores for each group (one line of code). What is the most striking thing that you notice?

Solution

```
ggplot(math, aes(x = course, y = phobia)) + geom_boxplot()
```

Boxplot `a` is just weird. The bar across the middle is actually at the top, and it has no bottom. (Noting something sensible like this is enough.) Boxplot `b` is hugely spread out.

The two groups have very different spreads, but that is not a problem as long as we remember to do the Welch-Satterthwaite test that does not assume equal spreads. This is the default in R, so we are good, at least with that.

By way of explanation: the course a scores have a number of values equal so that the 3rd quartile and the median are the name, and also that the first quartile and the minimum value are the same:

```
tmp = math %>% filter(course == "a")
tmp %>% count(phobia)
```

```
## # A tibble: 3 x 2
##   phobia     n
##    <int> <int>
## 1      6     2
## 2      7     2
## 3      8     1
```

```
summary(tmp)
```

```
##     course              phobia
##  Length:5           Min.   :6.0
##  Class :character   1st Qu.:6.0
##  Mode  :character   Median :7.0
##                     Mean   :6.8
##                     3rd Qu.:7.0
##                     Max.   :8.0
```

The phobia scores from course A are two 6's, two 7's and an 8. The median and third quartile are both 7, and the first quartile is the same as the lowest value, 6.

Technique note: I wanted to do two things with the phobia scores from course A: count up how many of each score, and show you what the five-number summary looks like. One pipe won't do this (the pipe "branches"), so I saved what I needed to use, before it branched, into a data frame `tmp` and then used `tmp` twice. Pipes are powerful, but not *all*-powerful.

(d) Explain briefly why a t-test would not be good for these data. (There are two things that you need to say.)

Solution

The easiest way to structure this is to ask yourself first what the t-test needs, and second whether you have it. The t-test assumes (approximately) normal data. The boxplot for group a doesn't even look symmetric, and the one for group b has an oddly asymmetric box. So I think the normality is in question here, and therefore another test would be better. (This is perhaps a bit glib of an answer, since there are only 5 values in each group, and so they can certainly look non-normal even if they actually are normal, but these values are all integers, so it is perhaps wise to be cautious.) We have the machinery to assess the normality for these, in one shot:

```
ggplot(math, aes(sample = phobia)) + stat_qq() +
    stat_qq_line() + facet_wrap(~course, ncol = 1,
    scales = "free")
```

I don't know what *you* make of those, but they both look pretty straight to me (and there are only five observations, so it's hard to judge). Course `b` maybe has a "hole" in it (three large values and two small ones). Maybe. I dunno. What I would *really* be worried about is outliers, and at least we don't have those. I mentioned in class that the t-tests are robust to non-normality. I ought to have expanded on that a bit: what really makes the t-test still behave itself with non-normality is when you have *large* samples, that is, when the Central Limit Theorem has had a chance to take hold. (That's what drives the normality not really being necessary in most cases.) But, even with small samples, exact normality doesn't matter so much. Here, we have two tiny samples, and so we have to insist a bit more, but only a bit more, on a more-or-less normal shape in each group. (It's kind of a double jeopardy in that the situation where normality matters most, namely with small samples, is where it's the hardest to judge, because samples of size 5 even from a normal distribution can look very non-normal.) But, the biggest threats to the t-test are big-time skewness and outliers, and we are not suffering too badly from those.

(e) Run a suitable test to compare the "typical" scores for the two courses. (You can use the version from a package rather than building your own.) What do you conclude?

Solution

This is an invite to use `smmr`:

```
library(smmr)
median_test(math, phobia, course)
```

```
## $table
##      above
## group above below
##     a     1     2
##     b     2     2
##
## $test
##        what     value
## 1 statistic 0.1944444
## 2        df 1.0000000
## 3   P-value 0.6592430
```

We are nowhere near rejecting equal medians; in fact, both courses are very close to 50–50 above and below the overall median.

If you look at the frequency table, you might be confused by something: there were 10 observations, but there are only $1+2+2+2=7$ in the table. This is because three of the observations were equal to the overall median, and had to be thrown away:

```
math %>% summarize(med = median(phobia))
```

```
## # A tibble: 1 x 1
##     med
##   <dbl>
## 1     7
```

```
math %>% count(phobia)
```

```
## # A tibble: 6 x 2
##   phobia     n
##    <int> <int>
## 1      1     1
## 2      2     1
## 3      6     2
## 4      7     3
## 5      8     2
## 6      9     1
```

The overall median was 7. Because the actual data were really discrete (the phobia scores could only be whole numbers), we risked losing a lot of our data when we did this test (and we didn't have much to begin with). The other thing to say is that with small sample sizes, the frequencies in the table have to be *very* lopsided for you to have a chance of rejecting the null. Something like this is what you'd need:

```
x = c(1, 1, 2, 6, 6, 6, 7, 8, 9, 10)
g = c(1, 1, 1, 1, 1, 2, 2, 2, 2, 2)
d = tibble(x, g)
median_test(d, x, g)
```

```
## $table
##      above
## group above below
##     1     0     3
##     2     4     0
##
## $test
##        what       value
## 1 statistic 7.000000000
## 2        df 1.000000000
## 3   P-value 0.008150972
```

I faked it up so that we had 10 observations, three of which were equal to the overall median. Of the rest, all the small ones were in group 1 and all the large ones were in group 2. This is lopsided enough to reject with, though, because of the small frequencies, there actually was a warning about "chi-squared approximation may be inaccurate".

There *was*, in the *chisq.test* inside *median-test*, but in *smmr* I didn't pass that warning back to the outside world.

8.6 *Medical instructions*

Do people understand medical instructions better at certain times of the day? In a study, students in a grade 12 class are randomly divided into two groups, A and B. All students see a video describing how to use an infant forehead thermometer. The students in Group A see the video at 8:30 am, while the students in Group B see the same video at 3:00 pm (on the same day). The next day, all the students are given a test on the material in the video (graded out of 100). The observed scores are in link (values separated by spaces).

(a) Read the data into R and display the (first ten) values.

Solution

Separated by spaces, so `read_delim`:

```
my_url = "http://www.utsc.utoronto.ca/~butler/c32/forehead.txt"
instr = read_delim(my_url, " ")
```

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

```
##   score = col_integer()
## )
```

```
instr
```

```
## # A tibble: 18 x 2
##    group score
##    <chr> <int>
##  1 A        88
##  2 A        89
##  3 A        79
##  4 A       100
##  5 A        98
##  6 A        89
##  7 A        65
##  8 A        94
##  9 A        95
## 10 A        91
## 11 B        87
## 12 B        69
## 13 B        78
## 14 B        79
## 15 B        83
## 16 B        90
## 17 B        85
## 18 B        58
```

(b) Obtain a suitable plot that will enable you to assess the assumptions for a two-sample t-test.

Solution

We need the values in each group to be approximately normally distributed. Side-by-side boxplots will do it:

```
ggplot(instr, aes(x = group, y = score)) + geom_boxplot()
```

or, if you like, separate (facetted) normal quantile plots, which I would do this way:

```
ggplot(instr, aes(sample = score)) + stat_qq() +
    stat_qq_line() + facet_wrap(~group, ncol = 1)
```

(c) Why might you have doubts about using a two-sample t-test here?

Solution

We are looking for non-normality in at least one of the groups. Here, both groups have an outlier at the low end that would be expected to pull the mean downward. I don't think there is left-skewness here, since there is no particular evidence of the high-end values being bunched up: the problem in both cases with normality is at the low end. One way or another, I'm expecting you to have noticed the outliers. Extra: last year, when I first drew the normal quantile plots, there was no `stat_qq_line`, so you had to imagine where

the line went if you did it this way. Without the line, these plots look somewhat curved, which would have pointed to left-skewness, but now we see that the lowest observation is too low, and maybe the second-lowest one as well, while the other observations are just fine.

(d) Run Mood's median test as in class (*without* using `smmr`). What do you conclude, in the context of the data? What recommendation would you make about the time of day to see the video? (You might get a warning about "chisquared approximation being incorrect", which you can ignore here.)

Solution

The overall median first:

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

```
## # A tibble: 1 x 1
##     med
##   <dbl>
## 1  87.5
```

87.5, which is not equal to any of the data values (they are all integers). This will avoid any issues with values-equal-to-median later.

Then, create and save a table of the value by group and above/below median. You can count either above or below (it comes out equivalently either way):

```
tab = with(instr, table(group, score > 87.5))
tab
```

```
##
## group FALSE TRUE
##     A     2    8
##     B     7    1
```

Then, chi-squared test for independence (the null) or association of some kind (the alternative). The `correct=F` is saying not to do Yates's correction, so that it would come out the same if you were doing it by hand ("observed minus expected, squared, divided by expected" and all that stuff).

```
chisq.test(tab, correct = F)
```

```
## Warning in chisq.test(tab, correct = F): Chi-
## squared approximation may be incorrect
```

```
##
##  Pearson's Chi-squared test
##
## data:  tab
## X-squared = 8.1, df = 1, p-value =
## 0.004427
```

The P-value is 0.0044, which is (much) smaller than 0.05, and therefore you can reject independence and conclude association: that is, whether a student scores above or below the median depends on which group they are in, or, that the median scores are different for the two groups.

The warning is because the expected frequencies are on the small side (if you have done this kind of problem by hand, you might remember something about "expected frequencies less than 5". This is that.) Here, the P-value is so small that we can afford to have it be inaccurate by a bit and still not affect the conclusion, so I think we are safe.

As for which group is better, well, the easiest way is to go back to your boxplots and see that the median for group A (8:30 am) is substantially higher than for group B (3:00pm). But you can also see it from your frequency table, if you displayed it:

```
tab
```

```
##
## group FALSE TRUE
##     A     2    8
##     B     7    1
```

Most of the people in the 8:30 am group scored above the median, and most of the people in the 3:00 pm group scored below the median. So the scores at 8:30 am were better overall.

As I write this, it is just after 3:00 pm and I am about to make myself a pot of tea!

Extra: about that `correct=F` thing. There was a point of view for a long time that when you are dealing with a 2×2 table, you can get better P-values by, before squaring "observed minus expected", taking 0.5 away from the absolute value of the difference. This is called Yates's correction. It is in the same spirit as the "continuity correction" that you might have encountered in the normal approximation to the binomial, where in the binomial you have to have a whole number of successes, but the normal allows fractional values as well. In about the 1960s, the usefulness of Yates's correction was shot down, for general contingency tables. There is, however, one

case where it *is* useful, and that is the case where the row totals and column totals are *fixed*.

What do I mean by that? Well, first let's look at a case where the totals are *not* all fixed. Consider a survey in which you want to see whether males and females agree or disagree on some burning issue of the day. You collect random samples of, say, 500 males and 500 females, and you count how many of them say Yes or No to your statement. You might get results like this:

To simplify things, we'll assume that everyone gave a Yes or a No answer, though you could add a column like *No answer* if you wanted to make it more realistic.

	Yes	No	Total
Males	197	303	500
Females	343	157	500
Total	540	460	1000

In this table, the row totals must be 500, because you asked this many males and this many females, and each one must have answered something. The column totals, however, are not fixed: you didn't know, ahead of time, that 540 people would answer "yes". That was just the way the data turned out, and if you did another survey with the same design, you'd probably get a different number of people saying "yes".

For another example, let's go back to Fisher (yes, *that* Fisher). A "lady" of his acquaintance claimed to be able, by drinking a cup of tea with milk and sugar in it, whether the milk or the sugar had been added first. Fisher, or, more likely, his housekeeper, prepared 8 cups of tea, 4 with milk first and 4 with sugar first. The lady knew that four of the cups had milk first, and her job was to say which four. The results might have been like this:

Actual			
	Milk first	sugar first	Total
Lady	Milk first	3	1	4
says	sugar first	1	3	4
Total		4	4	8

This time, all of the row totals and all of the column totals must be 4, regardless of what the lady thinks. Even if she thinks 5 of the cups of tea actually had milk first, she is going to pick 4 of them to say that they have milk first, since she knows there are only 4. In this case, all of the row and column totals are fixed at 4, and the right analysis is called Fisher's Exact Test, based on the hypergeometric distribution. In a 2×2 table like this one, there is only one "degree of freedom", since as soon as you specify one of the frequencies, say the number of cups where the lady said milk first and they actually

were milk first, you can work out the others. But, leaving that aside, the usual chi-squared analysis is a perfectly good approximation, especially if the frequencies are large, and especially if you use Yates's correction.

It is clear that Fisher must have been English, since he was able to get a publication out of drinking tea.

How does that apply to Mood's median test? Well, let's remind ourselves of the table we had:

```
tab
```

```
##
## group FALSE TRUE
##     A     2    8
##     B     7    1
```

We know how many students were in each group: 10 in group A and 8 in B. So the row totals are fixed. What about the columns? These are whether each observation was above or below the overall median. There were 18 observations altogether, so there *must* be 9 above and 9 below. So the column totals are fixed as well. All totals fixed, so we should be using Yates's correction. I didn't, because I wanted to keep things simple, but I should have done.

Except in the case of the previous problem, where there were multiple observations equal to the overall median. Which we ignore for the moment.

R's `chisq.test` by default *always* uses Yates's correction, and if you don't want it, you have to say `correct=F`. Which is why I have been doing so all through.

(e) Run Mood's median test on these data using my `smmr` package, and verify that you get the same answer.

Solution

Not much to it, since the data is already read in:

```
library(smmr)
median_test(instr, score, group)
```

```
## $table
##      above
## group above below
##     A     8     2
##     B     1     7
##
## $test
##        what       value
## 1 statistic 8.100000000
## 2        df 1.000000000
## 3   P-value 0.004426526
```

Identical, test statistic, degrees of freedom and P-value. The table of frequencies is also the same, just with columns rearranged. (In `smmr` I counted the number of values below the overall median, whereas in my build-it-yourself I counted the number of values above.)

9
Matched pairs t and sign test

```
library(tidyverse)
```

```
## -- Attaching packages ---- tidyverse 1.2.1 --
```

```
## v ggplot2 3.1.0     v purrr   0.2.5
## v tibble  1.4.2     v dplyr   0.7.8
## v tidyr   0.8.1     v stringr 1.3.1
## v readr   1.1.1     v forcats 0.3.0
```

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

```
library(smmr)
```

9.1 Measuring body fat

Athletes are concerned with measuring their body fat percentage. Two different methods are available: one using ultrasound, and the other using X-ray technology. We are interested in whether there is a difference in the mean body fat percentage as measured by these two methods, and if so, how big that difference is. Data on 16 athletes are at link.

(a) Explain briefly why a matched pairs analysis is more suitable for these data than a two-independent-samples analysis (using a two-sample t-test). You might find that looking at the data (clicking on the link) helps you figure this out.

Solution
The data file looks like this:

```
athlete xray ultrasound
```

```
1 5.00 4.75
2 7 3.75
3 9.25 9
4 12 11.75
5 17.25 17
6 29.5 27.5
7 5.5 6.5
8 6 6.75
9 8 8.75
10 8.5 9.5
11 9.25 9.5
12 11 12
13 12 12.25
14 14 15.5
15 17 18
16 18 18.25
```

The data are two measurements for each of the 16 athletes: that is, each athlete had their body fat percentage measured using *both* of the two methods. Extra: a two-sample t approach would be reasonable if one set of 16 athletes had been measured by X-ray and *another different* set of 16 athletes had been measured by ultrasound. (That is, if there had been 32 athletes in total, with each one randomly assigned to *one* of the measurement methods.) But that's not what happened. It is easy to measure one athlete's body fat percentage using both of the two methods, so a matched pairs design is easy to implement (as well as being better). If you use two independent samples (each athlete doing only one measurement method), you introduce an extra source of variability: athletes differ one from another in body fat, as well as differing possibly by measurement method. If you use a matched-pairs design, you remove the athlete-to-athlete differences, leaving only the differences due to measurement method.

(b) Read in the data and check that you have a sensible number of rows and columns.

Solution

This kind of thing. Since you looked at the data (didn't you?), you'll know that the values are separated by single spaces:

```r
myurl = "http://www.utsc.utoronto.ca/~butler/c32/bodyfat.txt"
bodyfat = read_delim(myurl, " ")
```

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

```
##   xray = col_double(),
##   ultrasound = col_double()
## )
```

```
bodyfat
```

```
## # A tibble: 16 x 3
##    athlete  xray ultrasound
##      <int> <dbl>      <dbl>
##  1       1  5          4.75
##  2       2  7          3.75
##  3       3  9.25       9
##  4       4 12         11.8
##  5       5 17.2       17
##  6       6 29.5       27.5
##  7       7  5.5        6.5
##  8       8  6          6.75
##  9       9  8          8.75
## 10      10  8.5        9.5
## 11      11  9.25       9.5
## 12      12 11         12
## 13      13 12         12.2
## 14      14 14         15.5
## 15      15 17         18
## 16      16 18         18.2
```

16 rows (athletes) and 3 columns, one for each measurement method and one labelling the athletes. All good.

Since 16 is not too much bigger than 10, I got the whole data frame here. (At least, I think that's the reason I got more than 10 rows.) In an R Notebook, you'll see the first ten rows as normal, with a button to click to see the other six.

(c) Carry out a suitable test to determine whether the means are the same or different. (At this point, obtain the R output including a P-value.)

Solution

Feed the two columns into `t.test` along with `paired=T`. This is a two-sided test, so we don't have to take any special steps for that. Note that we're back to the "old-fashioned" version of `t.test` that *does not* allow `data=`, so we have to go the `with` way:

```
with(bodyfat, t.test(xray, ultrasound, paired = T))
```

```
##
```

```
##  Paired t-test
##
## data:  xray and ultrasound
## t = -0.30801, df = 15, p-value = 0.7623
## alternative hypothesis: true difference in means is not equal to 0
## 95 percent confidence interval:
##  -0.7425068  0.5550068
## sample estimates:
## mean of the differences
##                -0.09375
```

(d) What do you conclude from the test?

Solution

The P-value of 0.7623 is not at all small, so there is no way we can reject the null hypothesis. There is no evidence of a difference in means; we can act as if the two methods produce the same mean body fat percentage. That is to say, on this evidence we can use either method, whichever one is cheaper or more convenient.

My hat stays on my head.

(e) Obtain a 95% confidence interval for the population mean difference. How is the interval consistent with your test?

Solution

You don't even need to do any more coding: the test was two-sided, so just pick the confidence interval off the output above: -0.74 to 0.56. The interval includes both positive and negative values (or, 0 is inside the interval), so the difference could go either way. This is entirely consistent with not being able to reject the null.

(f) Calculate the differences, and make a normal quantile plot of them. Is there any evidence that normality of differences fails? Explain briefly.

Solution

The smoothest way to do this is to use a pipeline: use a `mutate` to create the column of differences, and then pipe that into `ggplot`, omitting the data frame that would normally go first (the input data frame here is the new one with the differences in it, which doesn't have a name). I'll make a normal quantile plot in a moment, but if you haven't seen that yet, the plot to make is a histogram:

I learned yesterday that the Welsh word for *ironing* is *smwddio*, which seems weird until you say it out loud: it sounds like *smoothio*.

```
bodyfat %>% mutate(diff = xray - ultrasound) %>%
    ggplot(aes(x = diff)) + geom_histogram(bins = 6)
```

I don't know whether you'd call that "approximately normal" or not. We are in kind of a double-bind with this one: the sample size is small, so normality matters, but with a small sample, the data might not look very normal. It's kind of skewed right, but most of the evidence for the skewness is contained in those two observations with difference 2 and above, which is pretty flimsy evidence for anything. (In the normal quantile plot below, the suggestion is that those two observations really are a bit too large. It's easier to tell there.)

Below, I'm repeating the calculation of the differences, which is inefficient. If I'm going to draw two graphs of the differences, the right way is to calculate the differences *and save the data frame*, then use that new data frame twice. But you're probably only going to draw either the histogram or the normal quantile plot, not both, so you can use the appropriate one of my two bits of code. The normal quantile plot:

```
bodyfat %>% mutate(diff = xray - ultrasound) %>%
    ggplot(aes(sample = diff)) + stat_qq() + stat_qq_line()
```

This is showing a little evidence of skewness or outliers (depending on your point of view: either is good). The lowest and highest values are both too high, and the pattern of points on the plot is kind of curved (which would be evidence of skewness). Or you could say that the two highest values are too high, with the other values being more or less in line (that would be evidence of outliers at the upper end). I like outliers better than skewness, since those bottom-end points are not far off the line. I would also accept "no substantial problems", if you can make the case that those two highest points are not too far off the line. With only 16 observations as we have here, even truly normal data would stray off the line a bit.

As ever, your explanation is more important than your conclusion. Can you justify what you think?

If you took your differences the other way around, as `ultrasound` minus `xray`, your plot will also be the other way around, with the "outliers" at the bottom. That's good too.

Where this is going (which I didn't ask you) is whether or not we trust the result of the matched pairs test. I would say that the test is so far from being significant, and the failure of normality is not gross, that it is hard to imagine *any* alternative test coming up with a significant result. So I would be happy to trust this paired t-test.

9.2 *Throwing baseballs and softballs*

Can students throw a baseball farther than a softball? A statistics class, containing 24 students, went out to a football field to try to answer this question. Each student warmed up and then threw each type of ball as far as they could. The order of ball types was randomized: some students threw the baseball first, and some threw the softball first. (A softball is bigger than a baseball, so we might expect that

a softball would be harder to throw a long way than a baseball.) The data are in `http://www.utsc.utoronto.ca/~butler/c32/throw.txt` in three columns: the first is a number identifying the student, the second is the distance thrown with the baseball (in yards) and the third is the distance thrown with the softball (also in yards).

(a) Read the data into R. You'll need to supply some names to the columns.

Solution

This kind of thing:

```
myurl = "http://www.utsc.utoronto.ca/~butler/c32/throw.txt"
throws = read_delim(myurl, " ", col_names = c("student",
    "baseball", "softball"))
```

```
## Parsed with column specification:
## cols(
##   student = col_integer(),
##   baseball = col_integer(),
##   softball = col_integer()
## )
```

```
throws
```

```
## # A tibble: 24 x 3
##    student baseball softball
##      <int>    <int>    <int>
##  1       1       65       57
##  2       2       90       58
##  3       3       75       66
##  4       4       73       61
##  5       5       79       65
##  6       6       68       56
##  7       7       58       53
##  8       8       41       41
##  9       9       56       44
## 10      10       70       65
## # ... with 14 more rows
```

This is one of those times where we have to tell R what names to give the columns. Or you can put `col_names=F` and leave the columns called `X1, X2, X3` or whatever they end up as.

(b) Calculate a column of differences, baseball minus softball, *in* the data frame.

Solution

Add it to the data frame using `mutate`. Use the right-arrow assignment to create what I called `throws2` below, or put something like `throws2=` on the beginning of the line. Your choice.

```
throws2 <- throws %>% mutate(diff = baseball -
    softball)
```

(c) Carry out a sign test in R, testing the null hypothesis that the median difference is zero, against the alternative that it is greater than zero. Obtain a P-value. Your option whether you use `smmr` or not.

Solution

I think using `smmr` is way easier, so I'll do that first. There is even a shortcut in that the null median defaults to zero, which is exactly what we want here:

```
library(smmr)
sign_test(throws2, diff)
```

```
## $above_below
## below above
##     2    21
##
## $p_values
##   alternative      p_value
## 1       lower 9.999971e-01
## 2       upper 3.302097e-05
## 3   two-sided 6.604195e-05
```

We want, this time, the upper-tailed one-sided test, since we want to prove that students can throw a baseball a *longer* distance than a softball. Thus the P-value we want is 0.000033.

To build it yourself, you know the steps by now. First step is to count how many differences are greater and less than zero:

```
table(throws2$diff > 0)
```

```
##
## FALSE  TRUE
##     3    21
```

or

```
table(throws2$diff < 0)
```

```
##
## FALSE  TRUE
##    22     2
```

or, since we have things in a data frame,

```
throws2 %>% count(diff > 0)
```

```
## # A tibble: 2 x 2
##   `diff > 0`     n
##   <lgl>      <int>
## 1 FALSE          3
## 2 TRUE          21
```

or count those less than zero. I'd take any of those.

Note that these are *not all the same*. One of the differences is in fact exactly zero. The technically right thing to do with the zero difference is to throw it away (leaving 23 differences with 2 negative and 21 positive). I would take that, or 2 or 3 negative differences out of 24 (depending on whether you count "greater than zero" or "less than zero"). We hope that this won't make a material difference to the P-value; it'll make some difference, but won't (we hope) change the conclusion about whether to reject.

Second step is to get a P-value for whichever one of those you got, from the appropriate binomial distribution.

The P-value is the probability of getting 21 (or 22) positive differences out of 24 (or 23) or more, since this is the end of the distribution we should be at if the alternative hypothesis is correct. Thus any of these will get you a defensible P-value:

```
sum(dbinom(21:23, 23, 0.5))
```

```
## [1] 3.302097e-05
```

```
sum(dbinom(22:24, 24, 0.5))
```

```
## [1] 1.7941e-05
```

```
sum(dbinom(21:24, 24, 0.5))
```

```
## [1] 0.0001385808
```

```
sum(dbinom(0:2, 23, 0.5))
```

```
## [1] 3.302097e-05
```

```
sum(dbinom(0:2, 24, 0.5))
```

```
## [1] 1.7941e-05
```

```
sum(dbinom(0:3, 24, 0.5))
```

```
## [1] 0.0001385808
```

The first and fourth of those are the same as `smmr` (throwing away the exactly-median value).

As we hoped, there is no *material* difference here: there is no doubt with any of these possibilities that we will reject a median difference of zero in favour of a median difference greater than zero.

9.3 *Throwing baseballs and softballs, again*

Previously, you carried out a sign test to determine whether students could throw a baseball farther than a softball. This time, we will calculate a confidence interval for the median difference baseball minus softball, using the results of sign tests.

(a) Read the data into R from link, giving appropriate names to the columns, and add a column of differences.

Solution

I did it this way, combining the reading of the data with the calculation of the differences in *one* pipe:

```
myurl = "http://www.utsc.utoronto.ca/~butler/c32/throw.txt"
throws = read_delim(myurl, " ", col_names = c("student",
    "baseball", "softball")) %>% mutate(diff = baseball -
    softball)
```

```
## Parsed with column specification:
## cols(
##   student = col_integer(),
##   baseball = col_integer(),
##   softball = col_integer()
## )
```

```
throws
```

```
## # A tibble: 24 x 4
##    student baseball softball  diff
##      <int>    <int>    <int> <int>
##  1       1       65       57     8
##  2       2       90       58    32
##  3       3       75       66     9
##  4       4       73       61    12
##  5       5       79       65    14
##  6       6       68       56    12
```

```
##  7        7        58       53      5
##  8        8        41       41      0
##  9        9        56       44     12
## 10       10        70       65      5
## # ... with 14 more rows
```

(b) Use `smmr` to find a 95% confidence interval for the median difference.

Solution

`ci_median`, with 95% being the default confidence level:

```
ci_median(throws, diff)
```

```
## [1] 2.002930 8.999023
```

2 to 9. The ends of a CI for the median will be data values, which are all whole numbers, so round off that 8.999.

(c) What function in `smmr` will run a two-sided sign test and return only the P-value? Check that it works by testing whether the median difference for your data is zero or different from zero.

Solution

The rest of the way, we are trying to reproduce that confidence interval by finding it ourselves. The function is called `pval_sign`. If you haven't run into it before, in R Studio click on Packages, find `smmr`, and click on its name. This will bring up package help, which includes a list of all the functions in the package, along with a brief description of what each one does. (Clicking on a function name brings up the help for that function.) Let's check that it works properly by repeating the previous `sign_test` and verifying that `pval_sign` gives the same thing:

```
sign_test(throws, diff, 0)
```

```
## $above_below
## below above
##     2    21
##
## $p_values
##   alternative      p_value
## 1       lower 9.999971e-01
## 2       upper 3.302097e-05
## 3   two-sided 6.604195e-05
```

```
pval_sign(0, throws, diff)
```

```
## [1] 6.604195e-05
```

The P-values are the same (for the two-sided test) and both small, so the median difference is not zero.

(d) Based on your P-value, do you think 0 is inside the confidence interval or not? Explain briefly.

Solution

Absolutely not. The median difference is definitely not zero, so zero cannot be in the confidence interval. Our suspicion, from the one-sided test from earlier, is that the differences were mostly positive (people could throw a baseball farther than a softball, in most cases). So the confidence interval ought to contain only positive values. I ask this because it drives what happens below.

(e) Obtain a 95% confidence interval for the population median difference, baseball minus softball, using a trial-and-error procedure that determines whether a number of possible medians are inside or outside the CI.

Solution

I've given you a fair bit of freedom to tackle this as you wish. Anything that makes sense is good: whatever mixture of mindlessness, guesswork and cleverness that you want to employ. The most mindless way to try some values one at a time and see what you get, eg.:

```
pval_sign(1, throws, diff)
```

```
## [1] 0.001489639
```

```
pval_sign(5, throws, diff)
```

```
## [1] 1.168188
```

So median 1 is outside and median 5 is inside the 95% interval. Keep trying values until you've figured out where the lower and upper ends of the interval are: where the P-values cross from below 0.05 to above, or vice versa.

Something more intelligent is to make a long list of potential medians, and get the P-value for each of them, eg.:

```
d = tibble(my.med = seq(0, 20, 2))
d %>% mutate(pvals = map_dbl(my.med, ~pval_sign(.,
    throws, diff)))
```

```
## # A tibble: 11 x 2
##    my.med      pvals
```

```
##     <dbl>      <dbl>
##  1      0 0.0000660
##  2      2 0.0525
##  3      4 0.839
##  4      6 0.678
##  5      8 0.210
##  6     10 0.0227
##  7     12 0.00149
##  8     14 0.0000660
##  9     16 0.0000359
## 10     18 0.00000572
## 11     20 0.00000298
```

2 is just inside the interval, 8 is also inside, and 10 is outside. Some closer investigation:

```
d = tibble(my.med = seq(0, 2, 0.5))
d %>% mutate(pvals = map_dbl(my.med, ~pval_sign(.,
    throws, diff)))
```

```
## # A tibble: 5 x 2
##   my.med     pvals
##    <dbl>     <dbl>
## 1    0   0.0000660
## 2    0.5 0.000277
## 3    1   0.00149
## 4    1.5 0.0227
## 5    2   0.0525
```

The bottom end of the interval actually is 2, since 2 is inside and 1.5 is outside.

```
d = tibble(my.med = seq(8, 10, 0.5))
d %>% mutate(pvals = map_dbl(my.med, ~pval_sign(.,
    throws, diff)))
```

```
## # A tibble: 5 x 2
##   my.med  pvals
##    <dbl>  <dbl>
## 1    8   0.210
## 2    8.5 0.152
## 3    9   0.0525
## 4    9.5 0.0227
## 5   10   0.0227
```

The top end is 9, 9 being inside and 9.5 outside.

Since the data values are all whole numbers, I think this is accurate enough. The most sophisticated way is the "bisection" idea we saw before. We already have a kickoff for this, since we found, mindlessly, that 1 is outside the interval on the low end and 5 is inside, so the lower limit has to be between 1 and 5. Let's try halfway between, ie. 3:

```
pval_sign(3, throws, diff)
```

```
## [1] 0.3833103
```

Inside, so lower limit is between 1 and 3. This can be automated, thus:

```
lo = 1
hi = 3
while (abs(hi - lo) > 0.1) {
    try = (lo + hi)/2
    ptry = pval_sign(try, throws, diff)
    if (ptry > 0.05) {
        hi = try
    } else {
        lo = try
    }
}
c(lo, hi)
```

```
## [1] 1.9375 2.0000
```

The difficult bit is to decide whether the value `try` becomes the new `lo` or the new `hi`. If the P-value for the median of `try` is greater than 0.05, `try` is inside the interval, and it becomes the new `hi`; otherwise it's outside and becomes the new `lo`. Whatever the values are, `lo` is always outside the interval and `hi` is always inside, and they move closer and closer to each other.

At the other end of the interval, `lo` is inside and `hi` is outside, so there is a little switching around within the loop. For starting values, you can be fairly mindless: for example, we know that 5 is inside and something big like 20 must be outside:

```
lo = 5
hi = 20
while (abs(hi - lo) > 0.1) {
    try = (lo + hi)/2
    ptry = pval_sign(try, throws, diff)
    if (ptry > 0.05) {
```

```
        lo = try
    } else {
        hi = try
    }
}
c(lo, hi)
```

```
## [1] 8.984375 9.042969
```

The interval goes from 2 to (as calculated here) about 9. (This is apparently the same as `ci_median` in `smmr` got.) `ci_median` uses the bisection method with a smaller "tolerance" than we did, so its answer is more accurate. It looks as if the interval goes from 2 to 9: that is, students can throw a baseball on average between 2 and 9 feet further than they can throw a softball.

9.4 *Changes in salary*

A company is growing and would like to attract more employees. The company would like to advertise that salaries there are increasing. To do this, the company randomly samples 20 employees that have been working there since January 2016, and for each of these employees, records their salary in January 2016 and January 2017. The data, with salaries in thousands of dollars, are in link.

(a) Read the data into R and demonstrate that you have two salaries for each of 20 employees.

Solution

Looking at the file, we see that the values are separated by exactly one space:

```
my_url = "https://www.utsc.utoronto.ca/~butler/c32/salaryinc.txt"
salaries = read_delim(my_url, " ")
```

```
## Parsed with column specification:
## cols(
##   employee = col_character(),
##   jan2016 = col_double(),
##   jan2017 = col_double()
## )
```

```
salaries
```

```
## # A tibble: 20 x 3
##    employee jan2016 jan2017
##    <chr>      <dbl>   <dbl>
```

```
##  1 A           36      39.5
##  2 B           41.2    47
##  3 C           40      45
##  4 D           42.8    49
##  5 E           51      57.8
##  6 F           50.5    54
##  7 G           56      62
##  8 I           57.8    69.9
##  9 J           62      66.8
## 10 K           65.5    71
## 11 L           66      72
## 12 M           68.9    74
## 13 N           71      80
## 14 O           72.3    79
## 15 P           74.6    80
## 16 Q           77      83.1
## 17 R           79.9    82.5
## 18 S           81      92
## 19 T           83.2    85
## 20 U           90     101
```

There are 20 employees (rows), and two columns of salaries: for each employee in the data set, their salary in January 2016 and in January 2017 (thus, two salaries for each employee).

(b) To compare the salaries, explain briefly why a matched-pairs test would be better than a two-sample test.

Solution

A matched-pairs test would be better because we have two observations (salaries) for each subject (employee). A two-sample test would be appropriate if we had two separate sets of employees, one set with their salaries recorded in 2016 and the other with their salaries recorded in 2017. That is not what we have here. You can go after this either way: why a matched-pairs approach is appropriate, or why a two-sample approach is not (or a bit of both).

(c) Make a suitable graph to assess the assumptions for a matched-pairs t-test. What does your graph tell you?

Solution

This requires thought first before you do any coding (and this is the reason for this one being four points). What has to be at least approximately normally distributed is the set of *differences*, salary at one time point minus the salary at the other, for each employee. The individual salaries don't have to be normally distributed at all. We

don't have the differences here, so we have to calculate them first. The smoothest way is to make a pipeline:

```
salaries %>% mutate(diff = jan2017 - jan2016) %>%
    ggplot(aes(sample = diff)) + stat_qq() + stat_qq_line()
```

A couple of coding notes: (i) you can take the differences 2016 minus 2017 if you like (then they will tend to be negative), (ii) `ggplot` used in a pipeline like this does *not* have a data frame first (the data frame used is the nameless output from the `mutate`, with the differences in it).

Also, there's no problem doing the `mutate`, saving that, and then feeding the saved data frame into `ggplot`. If you find that clearer, go for it.

As for what I see: I think those points get a bit far from the line at the high and low ends: the high values are too high and the low values are too low, which is to say that we have outliers at both ends, or the distribution has long tails (either way of saying it is good).

The important conclusion here is whether these differences are normal *enough* to trust a matched pairs t-test here. We have a sample of size 20, so the central limit theorem will help us some, but you can reasonably say that these tails are too long and that we should not trust a matched-pairs t-test.

I actually wanted you to practice doing a matched-pairs t-test anyway, hence my comment in the next part, but the result is probably not *so* trustworthy.

(d) Carry out a suitable matched-pairs t-test on these data. (If you thought in the previous part that this was the wrong thing to do, do it anyway for the purposes of this assignment.) What do you conclude?

Solution

The company is trying to prove that salaries are *increasing* over time, so we need a one-sided alternative. Following through the procedure, even though you may not trust it much:

```
with(salaries, t.test(jan2016, jan2017, alternative = "less",
    paired = T))
```

```
##
##  Paired t-test
##
## data:  jan2016 and jan2017
## t = -10.092, df = 19, p-value =
## 2.271e-09
## alternative hypothesis: true difference in means is less than 0
## 95 percent confidence interval:
##       -Inf -5.125252
## sample estimates:
## mean of the differences
##                  -6.185
```

You could also have the years the other way around, in which case the alternative has to be the other way around as well:

```
with(salaries, t.test(jan2017, jan2016, alternative = "greater",
    paired = T))
```

```
##
##  Paired t-test
##
## data:  jan2017 and jan2016
## t = 10.092, df = 19, p-value =
## 2.271e-09
## alternative hypothesis: true difference in means is greater than 0
## 95 percent confidence interval:
##  5.125252      Inf
## sample estimates:
## mean of the differences
##                   6.185
```

Or, if you saved the data frame with the differences in it, do a one-sample test on those, again making sure that you get the `alternative` right. I didn't save it, so I'm calculating the differences again:

```
salaries %>% mutate(diff = jan2017 - jan2016) %>%
    with(., t.test(diff, mu = 0, alternative = "greater"))
```

```
##
##  One Sample t-test
##
## data:  diff
## t = 10.092, df = 19, p-value =
## 2.271e-09
## alternative hypothesis: true mean is greater than 0
## 95 percent confidence interval:
##  5.125252      Inf
## sample estimates:
## mean of x
##     6.185
```

Whichever way you do it, the P-value is the same 2.271×10^{-9}, which is a whole lot less than 0.05, so there is no doubt at all that salaries are increasing.

(Your intuition ought to have expected something like this, because everyone's 2017 salary appears to be greater than their 2016 salary.)

Extra: you might be feeling that we ought to be doing a matched-pairs sign test, which you could do this way:

```
library(smmr)
salaries %>% mutate(diff = jan2017 - jan2016) %>%
    sign_test(diff, 0)
```

```
## $above_below
## below above
##     0    20
##
## $p_values
##   alternative      p_value
## 1       lower 1.000000e+00
## 2       upper 9.536743e-07
## 3   two-sided 1.907349e-06
```

and then take the "upper" P-value, which is in the same ballpark as the one from the t-test. So the salaries really are increasing, whether you believe the t-test or not. And note that *every single employee's salary increased*.

(Again, the "missing" data frame in `sign_test` is the nameless one with the differences in it.)

(e) The company would like to estimate *how much* salaries are increasing, on average. Obtain some output that will enable the company to assess this, and tell the CEO which piece of the output they should look at.

Solution

A confidence interval. 95% is fine. As before, we have to run `t.test` again because we ran a one-sided test and a confidence interval for us is two-sided:

```
with(salaries, t.test(jan2017, jan2016, paired = T))
```

```
##
##  Paired t-test
##
## data:  jan2017 and jan2016
## t = 10.092, df = 19, p-value =
## 4.542e-09
## alternative hypothesis: true difference in means is not equal to 0
## 95 percent confidence interval:
##  4.902231 7.467769
## sample estimates:
## mean of the differences
##                   6.185
```

Between about $5,000 and about $7,500. This is what to tell the CEO.

9.5 *Body fat revisited*

Athletes are concerned with measuring their body fat percentage. Two different methods are available: one using ultrasound, and the other using X-ray technology. We are interested in whether there is a difference in the mean body fat percentage as measured by these two methods, and if so, how big that difference is. Data on 16 athletes are at link.

We saw this data set before.

(a) Read in the data again.

Solution

This kind of thing. Since you looked at the data (didn't you?), you'll know that the values are separated by single spaces:

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

```
## Parsed with column specification:
## cols(
##   athlete = col_integer(),
##   xray = col_double(),
##   ultrasound = col_double()
## )
```

```
bodyfat
```

```
## # A tibble: 16 x 3
##    athlete  xray ultrasound
##      <int> <dbl>      <dbl>
##  1       1  5          4.75
##  2       2  7          3.75
##  3       3  9.25       9
##  4       4 12         11.8
##  5       5 17.2       17
##  6       6 29.5       27.5
##  7       7  5.5        6.5
##  8       8  6          6.75
##  9       9  8          8.75
## 10      10  8.5        9.5
## 11      11  9.25       9.5
## 12      12 11         12
## 13      13 12         12.2
## 14      14 14         15.5
## 15      15 17         18
## 16      16 18         18.2
```

(b) Calculate the differences, and make a normal quantile plot of them. Is there any evidence that normality of differences fails? Explain briefly.

Solution

This is a good place to look ahead. We'll need the differences in two places, most likely: first for the normal quantile plot, and second for the matched-pairs sign test. So we should calculate and save them first:

```
bodyfat2 <- bodyfat %>% mutate(diff = xray - ultrasound)
```

I seem to be using a 2 on the end to name my dataframe-with-differences, but you can use whatever name you like.

Then, not forgetting to use the data frame that we just made:

```
ggplot(bodyfat2, aes(sample = diff)) + stat_qq() +
    stat_qq_line()
```

This is showing a little evidence of skewness or outliers (depending on your point of view: either is good). The lowest and highest values are both too high, and the pattern of points on the plot is kind of curved (which would be evidence of skewness). Or you could say that the two highest values are too high, with the other values being more or less in line (that would be evidence of outliers at the upper end). I like outliers better than skewness, since those bottom-end points are not far off the line. I would also accept "no substantial problems", if you can make the case that those two highest points are not too far off the line. With only 16 observations as we have here, even truly normal data would stray off the line a bit.

As ever, your explanation is more important than your conclusion. Can you justify what you think?

If you took your differences the other way around, as `ultrasound` minus `xray`, your plot will also be the other way around, with the "outliers" at the bottom. That's good too.

(c) Previously, we did a matched-pairs t-test for these data. In the light of your normal quantile plot, do you think that was a good idea? Explain briefly.

Solution

We are looking for the differences to be approximately normal, bearing in mind that we have a sample of size 16, which is not that large. Say what you think here; the points, if I were giving any here, would be for the way in which you support it. The comment I made before when we did a matched-pairs t-test was that the P-value was so large and non-significant that it was hard to imagine any other test giving a significant result. Another way of saying that is that I considered these differences to be "normal enough", given the circumstances. You might very well take a different view. You could

say that these differences are clearly not normal, and that the sample size of 16 is not large enough to get any substantial help from the Central Limit Theorem. From that point of view, running the t-test is clearly not advisable.

(d) Use the sign test appropriately to compare the two methods for measuring body fat. (Use `smmr` if you wish.) What do you conclude, as ever in the context of the data?

Solution

That means using a sign test to test the null hypothesis that the median difference is zero, against the alternative that it is not zero. (I don't see anything here to indicate that we are looking only for positive or only for negative differences, so I think two-sided is right. You need some reason to do a one-sided test, and there isn't one here.)

Remembering again to use the data frame that has the differences in it:

```
sign_test(bodyfat2, diff, 0)
```

```
## $above_below
## below above
##    10     6
##
## $p_values
##   alternative   p_value
## 1       lower 0.2272491
## 2       upper 0.8949432
## 3   two-sided 0.4544983
```

The two-sided P-value is 0.4545, so we are nowhere near rejecting the null hypothesis that the median difference is zero. There is no evidence that the two methods for measuring body fat show any difference on average.

The table of aboves and belows says that there were 6 positive differences and 10 negative ones. This is not far from an even split, so the lack of significance is entirely what we would expect.

Extra: this is the same conclusion that we drew the last time we looked at these data (with a matched-pairs t-test). That supports what I said then, which is that the t-test was so far from being significant, that it could be very wrong without changing the conclusion. That is what seems to have happened.

10
Normal quantile plots

```
library(tidyverse)
```

```
## -- Attaching packages ---- tidyverse 1.2.1 --
```

```
## v ggplot2 3.1.0     v purrr   0.2.5
## v tibble  1.4.2     v dplyr   0.7.8
## v tidyr   0.8.1     v stringr 1.3.1
## v readr   1.1.1     v forcats 0.3.0
```

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

10.1 Lengths of heliconia flowers

The tropical flower *Heliconia* is fertilized by hummingbirds, a different species for each variety of Heliconia. Over time, the lengths of the flowers and the form of the hummingbirds' beaks have evolved to match each other. The length of the Heliconia flower is therefore an important measurement. Does it have a normal distribution for each variety?

The data set at `http://www.utsc.utoronto.ca/~butler/c32/heliconia.csv` contains the lengths (in millimetres) of samples of flowers from each of three varieties of Heliconia: *bihai*, *caribaea* red, and *caribaea* yellow.

(a) Read the data into R. There are different numbers of length measurements for each variety. How does this show up in the data frame? (Look at all the rows, not just the first ten.)

Solution

The usual `read_csv`:

```
heliconia = read_csv("heliconia.csv")
```

```
## Parsed with column specification:
## cols(
##   bihai = col_double(),
##   caribaea_red = col_double(),
##   caribaea_yellow = col_double()
## )
```

I suggested to look at *all* the rows. Here's why:

```
heliconia %>% print(n = Inf)
```

```
## # A tibble: 23 x 3
##    bihai caribaea_red caribaea_yellow
##    <dbl>        <dbl>           <dbl>
##  1  47.1         41.9            36.8
##  2  46.8         42.0            37.0
##  3  46.8         41.9            36.5
##  4  47.1         43.1            36.1
##  5  46.7         41.5            36.0
##  6  47.4         41.7            35.4
##  7  46.4         39.8            38.1
##  8  46.6         40.6            37.1
##  9  48.1         39.6            35.2
## 10  48.3         42.2            36.8
## 11  48.2         40.7            36.7
## 12  50.3         37.9            35.7
## 13  50.1         39.2            36.0
## 14  46.3         37.4            34.6
## 15  46.9         38.2            34.6
## 16  48.4         38.1            NA
## 17  NA           38.1            NA
## 18  NA           38.0            NA
## 19  NA           38.8            NA
## 20  NA           38.2            NA
## 21  NA           38.9            NA
## 22  NA           37.8            NA
## 23  NA           38.0            NA
```

The varieties with fewer values have missings (NAs) attached to the end. This is because all the columns in a data frame have to have the same number of values. (The missings won't impact what we do below — we get a warning but not an error, and the plots are the same as they would be without the missings — but you might be aesthetically offended by them, in which case you can read what I do later on.)

(b) Make a normal quantile plot for the variety *bihai*.

Solution

There's a certain amount of repetitiveness here (that we work around later):

```
ggplot(heliconia, aes(sample = bihai)) + stat_qq() +
    stat_qq_line()
```

```
## Warning: Removed 7 rows containing non-finite
## values (stat_qq).
```

```
## Warning: Removed 7 rows containing non-finite
## values (stat_qq_line).
```

I'm saving the comments until we've seen all three.

(c) Make a normal quantile plot for the variety *Caribaea* red (note that the variable name in the data frame has an underscore in it).

Solution

Same idea again:

```
ggplot(heliconia, aes(sample = caribaea_red)) +
    stat_qq() + stat_qq_line()
```

(d) Make a normal quantile plot for the variety *Caribaea* yellow (this also has an underscore in it).

Solution

And, one more time:

```
ggplot(heliconia, aes(sample = caribaea_yellow)) +
    stat_qq() + stat_qq_line()
```

```
## Warning: Removed 8 rows containing non-finite
## values (stat_qq).
```

```
## Warning: Removed 8 rows containing non-finite
## values (stat_qq_line).
```

I did a lot of copying and pasting there.

(e) Which of the three varieties is closest to having a normal distribution? Explain (very) briefly.

Solution

Look at the three plots, and see which one stays closest to the line. To my mind, this is clearly the last one, *Caribaea* yellow. So your answer ought to be "*Caribaea* yellow, because the points are closest to the line". This, I would say, is acceptably close to normal, so using a t-test here would be fine. The answer "the last one" is not quite complete, because I asked you which *variety*, so your answer needs to name a variety.

(f) For each of the two other varieties, apart from the one you mentioned in the last part, describe briefly how their distributions fail to be normal.

Solution

Let's look at *bihai* first. I see this one as an almost classic curve: the points are above the line, then below, then above again. If you look at the data scale (y-axis), the points are too bunched up to be normal at the bottom, and too spread out at the top: that is, skewed to the *right*. You might also (reasonably) take the view that the points at the bottom are close to the line (not sure about the very smallest one, though), but the points at the top are farther away, so that what we have here is two outliers at the top. I'm OK with that. It's often difficult to distinguish between skewness and outliers (at the end of the long tail). What you conclude can often depend on how you look. We also had to look at the second plot, *caribaea* red. This is a rather strange one: the points veer away from the line at the ends, but look carefully: it is *not* outliers at both ends, but rather the points are *too bunched up* to be normal at both ends: that is, the distribution has *short tails* compared to the normal. It is something more like a uniform distribution, which has no tails at all, than a normal distribution, which won't have outliers but it *does* have some kind of tails. So, "short tails". Extra: that's all you needed, but I mentioned above that you might have been offended aesthetically by those missing values that were not really missing. Let's see if we can do this aesthetically. As you might expect, it uses several of the tools from the "tidyverse". First, tidy the data. The three columns of the data frame are all lengths, just lengths of different things, which need to be labelled. This is `gather` from `tidyr`.

```
heliconia.long = heliconia %>% gather(variety,
    length, bihai:caribaea_yellow, na.rm = T)
heliconia.long
```

```
## # A tibble: 54 x 2
##    variety length
##  * <chr>    <dbl>
```

```
##  1 bihai     47.1
##  2 bihai     46.8
##  3 bihai     46.8
##  4 bihai     47.1
##  5 bihai     46.7
##  6 bihai     47.4
##  7 bihai     46.4
##  8 bihai     46.6
##  9 bihai     48.1
## 10 bihai     48.3
## # ... with 44 more rows
```

This is now aesthetic as well as tidy: all those `NA` lines have gone (you can check that there are now $16 + 23 + 15 = 54$ rows of actual data, as there should be).

Now, how to get a normal quantile plot for each variety? This is `facet_wrap` on the end of the `ggplot` again.

```
ggplot(heliconia.long, aes(sample = length)) +
    stat_qq() + stat_qq_line() + facet_wrap(~variety,
    scale = "free")
```

These are a bit elongated vertically. The `scale="free"` allows a different vertical scale for each plot (otherwise there would be one vertical scale for all three plots); I decided that was best here since the typical lengths for the three varieties are different. *Caribaea* yellow is more or less straight, *bihai* has outliers (and may also be curved), *caribaea* red has that peculiar S-bend shape.

I didn't really like the vertical elongation. I'd rather have the plots be almost square, which they would be if we put them in three cells of a 2×2 grid. `facet_wrap` has `nrow` and `ncol` which you can use one

or both of to make this happen. This creates an array of plots with two columns and as many rows as needed:

```
ggplot(heliconia.long, aes(sample = length)) +
    stat_qq() + stat_qq_line() + facet_wrap(~variety,
    scale = "free", ncol = 2)
```

I think the square plots make it easier to see the shape of these: curved, S-bend, straightish. Almost the same code will get a histogram for each variety, which I'll also make squarish:

```
ggplot(heliconia.long, aes(x = length)) + geom_histogram(binwidth = 1) +
    facet_wrap(~variety, scale = "free", ncol = 2)
```

bihai has those two outliers, *caribaea* red has no tails to speak of (or you might say "it's bimodal", which would be another explanation of the pattern on the normal quantile plot), and *caribaea* yellow is shoulder-shruggingly normal (I looked at that and said, "well, I *guess*

If you have studied a thing called kurtosis, the fourth moment about the mean, you'll know that this measures *both* tail length *and* peakedness, so a short-tailed distribution also has a strong peak. Or, maybe, in this case, two strong peaks.

it's normal".) After you've looked at the normal quantile plots, you see what a crude tool a histogram is for assessing normality.

10.2 *Ferritin and normality*

In the lecture notes, we looked at some data on different athletes from the Australian Institute of Sport. This data set can be found at `http://www.utsc.utoronto.ca/~butler/c32/ais.txt`. Recall that the values are separated by *tabs*. In this question, we will assess one of the variables in the data set for normality.

(a)[1] Read the data set into R. (Only one point since you can copy from the lecture notes.)

Solution

`read_tsv` is the right thing:

```
my_url = "http://www.utsc.utoronto.ca/~butler/c32/ais.txt"
athletes = read_tsv(my_url)
```

```
## Parsed with column specification:
## cols(
##   Sex = col_character(),
##   Sport = col_character(),
##   RCC = col_double(),
##   WCC = col_double(),
##   Hc = col_double(),
##   Hg = col_double(),
##   Ferr = col_integer(),
##   BMI = col_double(),
##   SSF = col_double(),
##   `%Bfat` = col_double(),
##   LBM = col_double(),
##   Ht = col_double(),
##   Wt = col_double()
## )
```

```
athletes
```

```
## # A tibble: 202 x 13
##    Sex   Sport   RCC   WCC    Hc    Hg  Ferr
##    <chr> <chr> <dbl> <dbl> <dbl> <dbl> <int>
##  1 fema~ Netb~  4.56  13.3  42.2  13.6    20
##  2 fema~ Netb~  4.15   6    38    12.7    59
##  3 fema~ Netb~  4.16   7.6  37.5  12.3    22
##  4 fema~ Netb~  4.32   6.4  37.7  12.3    30
##  5 fema~ Netb~  4.06   5.8  38.7  12.8    78
```

```
##  6 fema~ Netb~  4.12   6.1  36.6  11.8    21
##  7 fema~ Netb~  4.17   5    37.4  12.7   109
##  8 fema~ Netb~  3.8    6.6  36.5  12.4   102
##  9 fema~ Netb~  3.96   5.5  36.3  12.4    71
## 10 fema~ Netb~  4.44   9.7  41.4  14.1    64
## # ... with 192 more rows, and 6 more
## #   variables: BMI <dbl>, SSF <dbl>,
## #   `%Bfat` <dbl>, LBM <dbl>, Ht <dbl>,
## #   Wt <dbl>
```

I listed the data to check that I had it right, but I didn't ask you to. (If you didn't have it right, that will show up soon enough.)

(b)[3] One of the variables, `Ferr`, is a measurement of Ferritin for each athlete. Obtain a normal quantile plot of the Ferritin values, for all the athletes together. What do you conclude about the shape of the distribution? Explain briefly.

Solution

As you would expect:

```
ggplot(athletes, aes(sample = Ferr)) + stat_qq() +
    stat_qq_line()
```

This is almost a classic right skew: the values are too bunched up at the bottom and too spread out at the top. The curved shape should be making you think "skewed" and then you can work out which way it's skewed.

(c)[3] It is possible that the shape you found in the previous part is because the athletes from all the different sports were mixed together. Use `ggplot` to obtain one normal quantile plot for each sport, collected together on one plot.

Solution

Your previous plot had all the sports mixed together. To that you add something that will put each sport in its own facet:

```
ggplot(athletes, aes(sample = Ferr)) + stat_qq() +
    stat_qq_line() + facet_wrap(~Sport)
```

(d)[2] Looking at the plots in the previous part, would you say that the Ferritin values for each sport individually have a more normal shape than they do for all the sports together? Explain briefly.

Solution

There are a couple of ways you can go, and as ever I'm looking mostly for consistency of argument. The two major directions you can go are (i) most of these plots are still curved the same way as the previous one, and (ii) they are mostly straighter than they were before. Possible lines of argument include that pretty much all of these plots are right-skewed still, with the same upward-opening curve. Pretty much the only one that doesn't is Gymnastics, for which there are only four observations, so you can't really tell. So, by this argument, Ferritin just *does* have a right-skewed distribution, and breaking things out by sport doesn't make much difference to that. Or, you could go another way and say that the plot of all the data together was *very* curved, and these plots are much less curved, that is to say, much less skewed. Some of them, such as basketball and netball, are almost straight, and they are almost normally distributed. Some of the distributions, such as track sprinting (`TSprnt`), are definitely still right-skewed, but not as seriously so as before. Decide what you think and then discuss how you see it.

11
Analysis of variance

```r
library(tidyverse)
```

```
## -- Attaching packages ---- tidyverse 1.2.1 --
```

```
## v ggplot2 3.1.0     v purrr   0.2.5
## v tibble  1.4.2     v dplyr   0.7.8
## v tidyr   0.8.1     v stringr 1.3.1
## v readr   1.1.1     v forcats 0.3.0
```

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

11.1 Movie ratings and lengths

Before a movie is shown in theatres, it receives a "rating" that says what kind of material it contains. link explains the categories, from G (suitable for children) to R (anyone under 17 must be accompanied by parent/guardian). In 2011, two students collected data on the length (in minutes) and the rating category, for 15 movies of each rating category, randomly chosen from all the movies released that year. The data are at link.

(a) Read the data into R, and display (some of) what you read in.

Solution

`read_csv`:

```r
my_url = "http://www.utsc.utoronto.ca/~butler/c32/movie-lengths.csv"
movies = read_csv(my_url)
```

```
## Parsed with column specification:
## cols(
##   length = col_integer(),
##   rating = col_character()
## )
```

```
movies
```

```
## # A tibble: 60 x 2
##    length rating
##     <int> <chr>
##  1     25 G
##  2     75 G
##  3     88 G
##  4     63 G
##  5     76 G
##  6     97 G
##  7     68 G
##  8     82 G
##  9     98 G
## 10     74 G
## # ... with 50 more rows
```

Something that looks like a length in minutes, and a rating.

(b) Count how many movies there are of each rating.

Solution

```
movies %>% count(rating)
```

```
## # A tibble: 4 x 2
##   rating     n
##   <chr>  <int>
## 1 G         15
## 2 PG        15
## 3 PG-13     15
## 4 R         15
```

Fifteen of each rating. (It's common to have the same number of observations in each group, but not necessary for a one-way ANOVA.)

(c) Carry out an ANOVA and a Tukey analysis (if warranted).

Solution
ANOVA first:

```
length.1 = aov(length ~ rating, data = movies)
summary(length.1)
```

```
##             Df Sum Sq Mean Sq F value
## rating       3  14624    4875   11.72
## Residuals   56  23295     416
```

```
##                Pr(>F)
## rating      4.59e-06 ***
## Residuals
## ---
## Signif. codes:
##   0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1  ' ' 1
```

This P-value is 0.00000459, which is way less than 0.05.

Having rejected the null (which said "all means equal"), we now need to do Tukey, thus:

```
TukeyHSD(length.1)
```

```
##   Tukey multiple comparisons of means
##     95% family-wise confidence level
##
## Fit: aov(formula = length ~ rating, data = movies)
##
## $rating
##                diff        lwr       upr
## PG-G      26.333333   6.613562 46.053104
## PG-13-G   42.800000  23.080229 62.519771
## R-G       30.600000  10.880229 50.319771
## PG-13-PG  16.466667  -3.253104 36.186438
## R-PG       4.266667 -15.453104 23.986438
## R-PG-13  -12.200000 -31.919771  7.519771
##              p adj
## PG-G     0.0044541
## PG-13-G  0.0000023
## R-G      0.0007379
## PG-13-PG 0.1327466
## R-PG     0.9397550
## R-PG-13  0.3660019
```

Cast your eye down the `p adj` column and look for the ones that are significant, here the first three. These are all comparisons with the G ("general") movies, which are shorter on average than the others (which are not significantly different from each other).

If you like, you can make a table of means to verify that:

```
movies %>% group_by(rating) %>% summarize(mean = mean(length))
```

```
## # A tibble: 4 x 2
##   rating  mean
##   <chr>  <dbl>
## 1 G       80.6
```

```
## 2 PG     107.
## 3 PG-13  123.
## 4 R      111.
```

When we do this problem in SAS, you'll see the Tukey get handled a different way, one that you might find more appealing.

(d) Make a graph to assess whether this ANOVA is trustworthy. Discuss your graph and its implications briefly.

Solution

The obvious graph is a boxplot:

```
ggplot(movies, aes(x = rating, y = length)) +
    geom_boxplot()
```

For ANOVA, we are looking for approximately normal distributions within each group and approximately equal spreads. Without the outliers, I would be more or less happy with that, but the G movies have a low outlier that would pull the mean down and the PG and PG-13 movies have outliers that would pull the mean up. So a comparison of means might make the differences look more significant than they should. Having said that, you could also say that the ANOVA is *very* significant, so even considering the effect of the outliers, the differences between G and the others are still likely to be significant.

Extra: the way to go if you don't trust the ANOVA is (as for the two-sample t) the Mood's median test. This applies to any number of groups, and works in the same way as before:

```
library(smmr)
median_test(movies, length, rating)
```

```
## $table
##        above
## group   above below
##   G         2    13
##   PG        7     7
##   PG-13    12     3
##   R         8     6
##
## $test
##        what        value
## 1 statistic 13.752380952
## 2        df  3.000000000
## 3   P-value  0.003262334
```

Still significant, though not quite as small a P-value as before (which echoes our thoughts about what the outliers might do to the means). If you look at the table above the test results, you see that the G movies are mostly shorter than the overall median, but now the PG-13 movies are mostly *longer*. So the picture is a little different.

Mood's median test does not naturally come with something like Tukey. What you can do is to do all the pairwise Mood's median tests, between each pair of groups, and then adjust to allow for your having done several tests at once. I thought this was generally useful enough that I put it into `smmr` under the name `pairwise_median_test`:

```
pairwise_median_test(movies, length, rating)
```

```
## # A tibble: 6 x 4
##   g1    g2      p_value adj_p_value
##   <chr> <chr>     <dbl>       <dbl>
## 1 G     PG    0.00799      0.0479
## 2 G     PG-13 0.0000590    0.000354
## 3 G     R     0.0106       0.0635
## 4 PG    PG-13 0.0106       0.0635
## 5 PG    R     0.715        4.29
## 6 PG-13 R     0.273        1.64
```

You can ignore those (adjusted) P-values rather stupidly bigger than 1. These are not significant.

There are two significant differences in median length: between G movies and the two flavours of PG movies. The G movies are significantly shorter (as you can tell from the boxplot), but the difference between G and R movies is no longer significant (a change from the regular ANOVA).

You may be puzzled by something in the boxplot: how is it that the G movies are significantly shorter than the PG movies, but not

significantly shorter than the R movies, *when the difference in medians between G and R movies is bigger*? In Tukey, if the difference in means is bigger, the P-value is smaller. The resolution to this puzzle, such as it is, is that Mood's median test is not directly comparing the medians of the groups (despite its name); it's counting values above and below a *joint* median, which might be a different story.

Actually, this doesn't always work if the sample sizes in each group are different. If you're comparing two small groups, it takes a *very large* difference in means to get a small P-value. But in this case the sample sizes are all the same.

11.2 *Deer and how much they eat*

Do adult deer eat different amounts of food at different times of the year? The data in link are the weights of food (in kilograms) consumed by randomly selected adult deer observed at different times of the year (in February, May, August and November). We will assume that these were different deer observed in the different months. (If the same animals had been observed at different times, we would have been in the domain of "repeated measures", which would require a different analysis, beyond the scope of this course.)

(a) Read the data into R, and calculate numbers of observations and the median amounts of food eaten each month.

Solution

The usual stuff for data values separated by spaces:

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

```
## Parsed with column specification:
## cols(
##   month = col_character(),
##   food = col_double()
## )
```

and then, recalling that `n()` is the handy way of getting the number of observations in each group:

```
deer %>% group_by(month) %>% summarize(n = n(),
    med = median(food))
```

```
## # A tibble: 4 x 3
##   month     n   med
##   <chr> <int> <dbl>
## 1 Aug       6  4.7
## 2 Feb       5  4.8
## 3 May       6  4.35
## 4 Nov       5  5.2
```

When you want the number of observations *plus* some other summaries, as here, the group-by and summarize idea is the way, using `n()` to get the number of observations in each group. `count` counts the number of observations per group when you *only* have grouping variables.

The medians differ a bit, but it's hard to judge without a sense of spread, which the boxplots (next) provide. November is a bit higher and May a bit lower.

(b) Make side-by-side boxplots of the amount of food eaten each month. Comment briefly on what you see.

Solution

```
ggplot(deer, aes(x = month, y = food)) + geom_boxplot()
```

This offers the suggestion that maybe November will be significantly higher than the rest and May significantly lower, or at least they will be significantly different from each other.

This is perhaps getting ahead of the game: we should be thinking about spread and shape. Bear in mind that there are only 5 or 6 observations in each group, so you won't be able to say much about normality. In any case, we are going to be doing a Mood's median test, so any lack of normality doesn't matter (eg. perhaps that 4.4 observation in August). Given the small sample sizes, I actually think the spreads are quite similar.

Another way of looking at the data, especially with these small sample sizes, is a "dot plot": instead of making a boxplot for each month, we plot the actual points for each month as if we were making a scatterplot:

```
ggplot(deer, aes(x = month, y = food)) + geom_point()
```

Wait a minute. There were five deer in February and six in August. Where did they go?

The problem is *overplotting*: more than one of the deer plotted in the same place on the plot, because the amounts of food eaten were only given to one decimal place and there were some duplicated values. One way to solve this is to randomly move the points around so that no two of them plot in the same place. This is called *jittering*, and is done like this:

```
ggplot(deer, aes(x = month, y = food)) + geom_jitter(width = 0,
    height = 0.05)
```

Now you see all the deer, and you can see that two pairs of points in August and one pair of points in February are close enough on the jittered plot that they would have been the same to one decimal place.

I wanted to keep the points above the months they belong to, so I only allowed vertical jitter (that's the `width` and `height` in the

geom_jitter; the width is zero so there is no horizontal jittering). If you like, you can colour the months; it's up to you whether you think that's making the plot easier to read, or is overkill (see my point on the facetted plots on the 2017 midterm).

This way you see the whole distribution for each month. Normally it's nicer to see the summary made by the boxplots, but here there are not very many points. The value of 4.4 in August does look quite a bit lower than the rest, but the other months look believably normal given the small sample sizes. I don't know about equal spreads (November looks more spread out), but normality looks believable. Maybe this is the kind of situation in which Welch's ANOVA is a good idea. (If you believe that the normality-with-unequal-spreads is a reasonable assumption to make, then the Welch ANOVA will be more powerful than the Mood's median test, and so should be preferred.)

(c) Run a Mood's median test as in lecture (ie. not using smmr). What do you conclude, in the context of the data?

Solution

To give you some practice with the mechanics, first find the overall median:

```
deer %>% summarize(med = median(food))
```

```
## # A tibble: 1 x 1
##     med
##   <dbl>
## 1   4.7
```

or

```
median(deer$food)
```

```
## [1] 4.7
```

I like the first way because it's the same idea as we did before, just not differentiating by month. I think there are some observations exactly equal to the median, which will mess things up later:

```
deer %>% filter(food == 4.7)
```

```
## # A tibble: 4 x 2
##   month  food
##   <chr> <dbl>
## 1 Feb     4.7
## 2 Feb     4.7
## 3 Aug     4.7
## 4 Aug     4.7
```

There are, two in February and two in August.

Next, make (and save) a table of the observations within each month that are above and below this median:

```
tab1 = with(deer, table(month, food < 4.7))
tab1
```

```
##
## month FALSE TRUE
##   Aug     4    2
##   Feb     5    0
##   May     0    6
##   Nov     5    0
```

or

```
tab2 = with(deer, table(month, food > 4.7))
tab2
```

```
##
## month FALSE TRUE
##   Aug     4    2
##   Feb     2    3
##   May     6    0
##   Nov     0    5
```

Either of these is good, but note that they are different. Two of the February observations (the ones that were exactly 4.7) have "switched sides", and (look carefully) two of the August ones also. Hence the test results will be different, and `smmr` (later) will give different results again:

```
chisq.test(tab1, correct = F)
```

```
## Warning in chisq.test(tab1, correct = F):
## Chi-squared approximation may be incorrect
```

```
##
##  Pearson's Chi-squared test
##
## data:  tab1
## X-squared = 16.238, df = 3, p-value =
## 0.001013
```

```
chisq.test(tab2, correct = F)
```

```
## Warning in chisq.test(tab2, correct = F):
## Chi-squared approximation may be incorrect
```

```
##
##  Pearson's Chi-squared test
##
## data:  tab2
## X-squared = 11.782, df = 3, p-value =
## 0.008168
```

The warnings are because of the small frequencies. If you've done these by hand before (which you will have if you took PSYC08), you'll remember that thing about "expected frequencies less than 5". This is that. It means "don't take those P-values *too* seriously."

The P-values are different, but they are both clearly significant, so the median amounts of food eaten in the different months are not all the same. (This is the same "there are differences" that you get from an ANOVA, which you would follow up with Tukey.) Despite the injunction not to take the P-values too seriously, I think these are small enough that they could be off by a bit without affecting the conclusion.

The first table came out with a smaller P-value because it looked more extreme: all of the February measurements were taken as higher than the overall median (since we were counting "strictly less" and "the rest"). In the second table, the February measurements look more evenly split, so the overall P-value is not quite so small.

You can make a guess as to what `smmr` will come out with (next), since it throws away any data values exactly equal to the median.

(d) Run a Mood's median test using `smmr`, and compare the results with the previous part.

Solution

Off we go:

```r
library(smmr)
median_test(deer, food, month)
```

```
## $table
##      above
## group above below
##   Aug     2     2
##   Feb     3     0
##   May     0     6
##   Nov     5     0
##
## $test
##        what        value
## 1 statistic 13.950000000
```

```
## 2        df  3.000000000
## 3   P-value  0.002974007
```

The P-value came out in between the other two, but the conclusion is the same all three ways: the months are not all the same in terms of median food eaten. The researchers can then go ahead and try to figure out *why* the animals eat different amounts in the different months.

You might be wondering how you could get rid of the equal-to-median values in the build-it-yourself way. This is `filter` from `dplyr`, which you use first:

```
deer2 = deer %>% filter(food != 4.7)
tab3 = with(deer2, table(month, food < 4.7))
tab3
```

```
##
## month FALSE TRUE
##   Aug     2    2
##   Feb     3    0
##   May     0    6
##   Nov     5    0
```

```
chisq.test(tab3)
```

```
## Warning in chisq.test(tab3): Chi-squared
## approximation may be incorrect
```

```
##
##  Pearson's Chi-squared test
##
## data:  tab3
## X-squared = 13.95, df = 3, p-value =
## 0.002974
```

which is exactly what `smmr` does, so the answer is identical. How would an ANOVA come out here? My guess is, very similarly:

```
deer.1 = aov(food ~ month, data = deer)
summary(deer.1)
```

```
##             Df Sum Sq Mean Sq F value
## month        3 2.3065  0.7688   22.08
## Residuals   18 0.6267  0.0348
##               Pr(>F)
## month       2.94e-06 ***
## Residuals
```

The computer scientists among you will note that I should not use equals or not-equals to compare a decimal floating-point number, since decimal numbers are not represented exactly in the computer. R, however, is ahead of us here, since when you try to do food not equal to 4.7, it tests whether food is more than a small distance away from 4.7, which is the right way to do it. In R, therefore, code like my *food != 4.7* does exactly what I want, but in something like C, it *does not*, and you have to be more careful: *abs(food-4.7)>1e-8*, or something like that. The small number *1e-8* is typically equal to **machine epsilon**, the smallest number on a computer that is distinguishable from zero.

```
## ---
## Signif. codes:
##   0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1  ' ' 1
```

```
TukeyHSD(deer.1)
```

```
##   Tukey multiple comparisons of means
##     95% family-wise confidence level
##
## Fit: aov(formula = food ~ month, data = deer)
##
## $month
##               diff         lwr        upr
## Feb-Aug  0.1533333 -0.16599282  0.4726595
## May-Aug -0.3333333 -0.63779887 -0.0288678
## Nov-Aug  0.5733333  0.25400718  0.8926595
## May-Feb -0.4866667 -0.80599282 -0.1673405
## Nov-Feb  0.4200000  0.08647471  0.7535253
## Nov-May  0.9066667  0.58734052  1.2259928
##             p adj
## Feb-Aug 0.5405724
## May-Aug 0.0290758
## Nov-Aug 0.0004209
## May-Feb 0.0021859
## Nov-Feb 0.0109631
## Nov-May 0.0000013
```

The conclusion is the same, but the P-value on the F-test is much smaller. I think this is because the F-test uses the actual values, rather than just whether they are bigger or smaller than 4.7. The Tukey says that all the months are different in terms of (now) mean, except for February and August, which were those two very similar ones on the boxplot.

(e) How is it that Mood's median test does not completely answer the question you really want to answer? How might you get an answer to the question you *really* want answered? Explain briefly, and obtain the answer you *really* want, discussing your results briefly.

Solution

That's rather a lot, so let's take those things one at a time.

Most of these parts are old from assignment questions that I actually asked a previous class to do, but not this part. I added it later.

Mood's median test is really like the F-test in ANOVA: it's testing the null hypothesis that the groups (months) all have the same median (of food eaten), against the alternative that the null is not true. We rejected this null, but we don't know which months differ

significantly from which. To resolve this in ANOVA, we do Tukey (or Games-Howell if we did the Welch ANOVA). The corresponding thing here is to do all the possible two-group Mood tests on all the pairs of groups, and, after adjusting for doing (here) six tests at once, look at the adjusted P-values to see how the months differ in terms of food eaten.

This is accomplished in `smmr` via `pairwise_median_test`, thus:

```
pairwise_median_test(deer, food, month)
```

```
## # A tibble: 6 x 4
##   g1    g2    p_value adj_p_value
##   <chr> <chr>   <dbl>       <dbl>
## 1 Aug   Feb   0.147       0.884
## 2 Aug   May   0.0209      0.126
## 3 Aug   Nov   0.00270     0.0162
## 4 Feb   May   0.00157     0.00939
## 5 Feb   Nov   0.0578      0.347
## 6 May   Nov   0.00157     0.00939
```

This compares each month with each other month. Looking at the last column, there are only three significant differences: August-November, February-May and May-November. Going back to the table of medians we made in (a), November is significantly higher (in terms of median food eaten) than August and May (but not February), and February is significantly higher than May. The other differences are not big enough to be significant.

Extra: Pairwise median tests done this way are not likely to be very sensitive (that is, powerful), for a couple of reasons: (i) the usual one that the median tests don't use the data very efficiently, and (ii) the way I go from the unadjusted to the adjusted P-values is via Bonferroni (here, multiply the P-values by 6), which is known to be safe but conservative. This is why the Tukey produced more significant differences among the months than the pairwise median tests did.

11.3 *Movie ratings again*

This question again uses the movie rating data at link.

(a) Read the data into R and obtain the number of movies of each rating and the *median* length of movies of each rating.

Solution

Reading in is as in the other question using these data (just copy your code, or mine). No credit for that, since you've done it before.

```
my_url = "http://www.utsc.utoronto.ca/~butler/c32/movie-lengths.csv"
movies = read_csv(my_url)
```

```
## Parsed with column specification:
## cols(
##   length = col_integer(),
##   rating = col_character()
## )
```

```
movies
```

```
## # A tibble: 60 x 2
##    length rating
##     <int> <chr>
##  1     25 G
##  2     75 G
##  3     88 G
##  4     63 G
##  5     76 G
##  6     97 G
##  7     68 G
##  8     82 G
##  9     98 G
## 10     74 G
## # ... with 50 more rows
```

Now, the actual for-credit part, which is a `group_by` and `summarize`:

```
movies %>% group_by(rating) %>% summarize(count = n(),
    med = median(length))
```

```
## # A tibble: 4 x 3
##   rating count   med
##   <chr>  <int> <int>
## 1 G         15    82
## 2 PG        15   100
## 3 PG-13     15   117
## 4 R         15   103
```

The G movies have a smaller median than the others, but also the PG-13 movies seem to be longer on average (not what we found before).

(b) Obtain a suitable graph that assesses the assumptions for ANOVA. Why do you think it is not reasonable to run ANOVA here? Explain briefly.

Solution

The graph would seem to be a boxplot, side by side for each group:

```
ggplot(movies, aes(x = rating, y = length)) +
    geom_boxplot()
```

We are looking for approximate normal distributions with approximately equal spreads, which I don't think we have: there are outliers, at the low end for G movies, and at the high end for PG and PG-13 movies. Also, you might observe that the distribution of lengths for R movies is skewed to the right. (Noting either the outliers or skewness as a reason for not believing normality is enough, since all we need is *one* way that normality fails.)

I think the spreads (as measured by the interquartile ranges) are acceptably similar, but since we have rejected normality, it is a bit late for that.

So I think it is far from reasonable to run an ANOVA here. In my opinion 15 observations in each group is not enough to gain much from the Central Limit Theorem either.

Extra: since part of the assumption for ANOVA is (approximate) normality, it would also be entirely reasonable to make normal quantile plots, one for each movie type, facetted. Remember the process: you pretend that you are making a normal quantile plot for all the data together, regardless of group, and then at the last minute, you throw in a `facet_wrap`. I've written the code out on three lines, so that you can see the pieces: the "what to plot", then the normal quantile plot part, then the facetting:

```
ggplot(movies, aes(sample = length)) + stat_qq() +
    stat_qq_line() + facet_wrap(~rating)
```

Since there are four movie ratings, `facet_wrap` has arranged them into a 2×2 grid, which satisfyingly means that each normal quantile plot is more or less square and thus easy to interpret.

The principal problem unveiled by these plots is outliers. It looks as if the G movies have one low outlier, the PG movies have two high outliers, the PG-13 movies have one or maybe three high outliers (depending on how you count them), and the R movies have none. Another way to look at the last two is you could call them curved, with too much bunching up at the bottom and (on PG-13) too much spread-out-ness at the top, indicating right-skewed distributions. The distribution of lengths of the R-rated movies is too bunched up at the bottom, but as you would expect for a normal at the top. The R movies show the right-skewedness in an odd way: usually this skewness shows up by having too many high values, but this time it's having too *few low* values.

The assumption for ANOVA is that all four of these are at least approximately normal (with the same spread). We found problems with the normality on at least three of them, so we definitely have doubts about trusting ANOVA here.

I could have used `scales=free` here to get a separate y-axis for each plot, but since the y-axis is movie length each time, and all four groups would be expected to have at least roughly similar movie lengths, I left it as it was. (The other advantage of leaving the scales the same is that you can compare spread by comparing the slopes of the lines on these graphs; since the lines connect the observed and theoretical quartiles, a steeper slope means a larger IQR. Here, the R line is steepest and the PG line is flattest. Compare this with the spreads of the boxplots.)

Extra extra: if you want, you can compare the normal quantile plots with the boxplots to see whether you get the same conclusion

from both. For the G movies, the low outlier shows up both ways, and the rest of the distribution is at least more or less normal. For the PG movies, I'd say the distribution is basically normal except for the highest two values (on both plots). For the PG-13 movies, only the highest value shows up as an outlier, but the next two apparent outliers on the normal quantile plot are at the upper end of the long upper whisker, so the boxplot is saying "right-skewed with one upper outlier" rather than "three upper outliers". The distribution of the R movies is skewed right, with the bunching at the bottom showing up as the very small lower whisker.

The boxplots and the normal quantile plots are basically telling the same story in each case, but they are doing it in a slightly different way.

(c) Run a Mood's median test (use `smmr` if you like). What do you conclude, in the context of the data?

Solution

The smart way is to use `smmr`, since it is much easier:

```
library(smmr)
median_test(movies, length, rating)
```

```
## $table
##        above
## group   above below
##   G         2    13
##   PG        7     7
##   PG-13    12     3
##   R         8     6
##
## $test
##        what        value
## 1 statistic 13.752380952
## 2        df  3.000000000
## 3   P-value  0.003262334
```

The movies do not all have the same median length, or at least one of the rating types has movies of different median length from the others. Or something equivalent to that. It's the same conclusion as for ANOVA, only with medians instead of means.

You can speculate about why the test came out significant. My guess is that the G movies are shorter than average, and that the PG-13 movies are longer than average. (We had the first conclusion before, but not the second. This is where medians are different from means.)

The easiest way to see which movie types really differ in length from which is to do all the pairwise median tests, which is in `smmr` thus:

```
pairwise_median_test(movies, length, rating)
```

```
## # A tibble: 6 x 4
##   g1    g2      p_value adj_p_value
##   <chr> <chr>     <dbl>       <dbl>
## 1 G     PG    0.00799      0.0479
## 2 G     PG-13 0.0000590    0.000354
## 3 G     R     0.0106       0.0635
## 4 PG    PG-13 0.0106       0.0635
## 5 PG    R     0.715        4.29
## 6 PG-13 R     0.273        1.64
```

The inputs for this are the same ones in the same order as for `median_test`. (A design decision on my part, since otherwise *I* would never have been able to remember how to run these!) Only the first two of these are significant (look in the last column). We can remind ourselves of the sample medians:

```
movies %>% group_by(rating) %>% summarize(count = n(),
    med = median(length))
```

```
## # A tibble: 4 x 3
##   rating count   med
##   <chr>  <int> <int>
## 1 G         15    82
## 2 PG        15   100
## 3 PG-13     15   117
## 4 R         15   103
```

The G movies are significantly shorter than the PG and PG-13 movies, but not quite significantly different from the R movies. This is a little odd, since the difference in sample medians between G and PG, significant, is *less* than for G and R (not significant). There are several Extras here, which you can skip if you don't care about the background. First, we can do the median test by hand: This has about four steps: (i) find the median of all the data, (ii) make a table tabulating the number of values above and below the overall median for each group, (iii) test the table for association, (iv) draw a conclusion. Thus (i):

```
median(movies$length)
```

```
## [1] 100
```

or

```
movies %>% summarize(med = median(length))
```

```
## # A tibble: 1 x 1
##     med
##   <dbl>
## 1   100
```

or store it in a variable, and then (ii):

```
tab1 = with(movies, table(length < 100, rating))
tab1
```

```
##        rating
##          G PG PG-13  R
##   FALSE  2  8    12  9
##   TRUE  13  7     3  6
```

or

```
tab2 = with(movies, table(length > 100, rating))
tab2
```

```
##        rating
##          G PG PG-13  R
##   FALSE 13  8     3  7
##   TRUE   2  7    12  8
```

These differ because there are evidently some movies of length exactly 100 minutes, and it matters whether you count $<$ and $\geq$ (as in `tab1`) or $>$ and *le* (`tab2`). Either is good.

Was I right about movies of length exactly 100 minutes?

```
movies %>% filter(length == 100)
```

```
## # A tibble: 2 x 2
##   length rating
##    <int> <chr>
## 1    100 PG
## 2    100 R
```

One PG and one R. It makes a difference to the R movies, but if you look carefully, it makes a difference to the PG movies as well, because the False and True switch roles between `tab1` and `tab2` (compare the G movies, for instance). You need to store your table in a variable because it has to get passed on to `chisq.test` below, (iii):

```
chisq.test(tab1, correct = F)
```

```
##
##  Pearson's Chi-squared test
##
## data:  tab1
## X-squared = 14.082, df = 3, p-value =
## 0.002795
```

or

```
chisq.test(tab2, correct = F)
```

```
##
##  Pearson's Chi-squared test
##
## data:  tab2
## X-squared = 13.548, df = 3, p-value =
## 0.003589
```

Either is correct, or, actually, without the `correct=F`.

see discussion elsewhere about Yates' Correction and fixed margins.

The conclusion (iv) is the same either way: the null of no association is clearly rejected (with a P-value of 0.0028 or 0.0036 as appropriate), and therefore whether a movie is longer or shorter than median length depends on what rating it has: that is, the median lengths do differ among the ratings. The same conclusion, in other words, as the F-test gave, though with not quite such a small P-value.

Second, you might be curious about how we might do something like Tukey having found some significant differences (that is, what's lurking in the background of `pairwise_median_test`).

Let's first suppose we are comparing G and PG movies. We need to pull out just those, and then compare them using `smmr`. Because the first input to `median_test` is a data frame, it fits neatly into a pipe (with the data frame omitted):

```
movies %>% filter(rating == "G" | rating == "PG") %>%
    median_test(length, rating)
```

```
## $table
##      above
## group above below
##    G      4    11
##    PG    10     3
##
## $test
##        what       value
## 1 statistic 7.035897436
## 2        df 1.000000000
## 3   P-value 0.007989183
```

We're going to be doing this about six times — $\binom{4}{2} = 6$ choices of two rating groups to compare out of the four — so we should have a function to do it. I think the input to the function should be a data frame that has a column called `rating`, and two names of ratings to compare:

```
comp2 = function(rat_1, rat_2, d) {
    d %>% filter(rating == rat_1 | rating == rat_2) %>%
        median_test(length, rating)
}
```

The way I wrote this function is that you have to specify the movie ratings in quotes. It is *possible* to write it in such a way that you input them without quotes, `tidyverse` style, but that gets into "non-standard evaluation" and `enquo()` and `!!`, which (i) I have to look up every time I want to do it, and (ii) I am feeling that the effort involved in explaining it to you is going to exceed the benefit you will gain from it. I mastered it enough to make it work in `smmr` (note that you specify column names without quotes there). There are tutorials on this kind of thing if you're interested.

Anyway, testing:

```
comp2("G", "PG", movies)
```

```
## $table
##      above
## group above below
##    G      4    11
##    PG    10     3
##
## $test
##        what       value
## 1 statistic 7.035897436
## 2        df 1.000000000
## 3   P-value 0.007989183
```

That works, but I really only want to pick out the P-value, which is in the list item `test` in the column `value`, the third entry. So let's rewrite the function to return just that:

```
comp2 = function(rat_1, rat_2, d) {
    d %>% filter(rating == rat_1 | rating == rat_2) %>%
        median_test(length, rating) %>% pluck("test",
        "value", 3)
}
comp2("G", "PG", movies)
```

```
## [1] 0.007989183
```

Gosh.

What `median_test` returns is an R `list` that has two things in it, one called `table` and one called `test`. The thing called `test` is a data frame with a column called `value` that contains the P-values. The third of these is the two-sided P-value that we want.

You might not have seen `pluck` before. This is a way of getting things out of complicated data structures. This one takes the output from `median_test` and from it grabs the piece called `test`. This is a data frame. Next, we want the column called `value`, and from that we want the third row. These are specified one after the other to `pluck` and it pulls out the right thing.

So now our function returns just the P-value.

I have to say that it took me several goes and some playing around in R Studio to sort this one out. Once I thought I understood `pluck`, I wondered why my function was not returning a value. And then I realized that I was saving the value inside the function and not returning it. Ooops. The nice thing about `pluck` is that I can put it on the end of the pipeline and and it will pull out (and return) whatever I want it to.

Let's grab a hold of the different rating groups we have:

```
the_ratings = unique(movies$rating)
the_ratings
```

```
## [1] "G"     "PG-13" "PG"    "R"
```

The Pythonisti among you will know how to finish this off: do a loop-inside-a-loop over the rating groups, and get the P-value for each pair. You can do that in R, if you must. It's not pretty at all, but it works:

```
ii = character(0)
jj = character(0)
pp = numeric(0)
for (i in the_ratings) {
    for (j in the_ratings) {
        pval = comp2(i, j, movies)
        ii = c(ii, i)
        jj = c(jj, j)
        pp = c(pp, pval)
    }
}
tibble(ii, jj, pp)
```

```
## # A tibble: 16 x 3
##    ii    jj           pp
##    <chr> <chr>     <dbl>
##  1 G     G     1
##  2 G     PG-13 0.0000590
##  3 G     PG    0.00799
##  4 G     R     0.0106
##  5 PG-13 G     0.0000590
##  6 PG-13 PG-13 1
##  7 PG-13 PG    0.0106
##  8 PG-13 R     0.273
##  9 PG    G     0.00799
## 10 PG    PG-13 0.0106
## 11 PG    PG    1
## 12 PG    R     0.715
## 13 R     G     0.0106
## 14 R     PG-13 0.273
## 15 R     PG    0.715
## 16 R     R     1
```

This is a lot of fiddling about, since you have to initialize three vectors, and then update them every time through the loop. It's hard to read, because the actual business part of the loop is the calculation of the P-value, and that's almost hidden by all the book-keeping. (It's also slow and inefficient, though the slowness doesn't matter too much here since it's not a very big problem.)

Let's try another way:

```
crossing(first = the_ratings, second = the_ratings)
```

```
## # A tibble: 16 x 2
##    first second
##    <chr> <chr>
##  1 G     G
##  2 G     PG
##  3 G     PG-13
##  4 G     R
##  5 PG    G
##  6 PG    PG
##  7 PG    PG-13
##  8 PG    R
##  9 PG-13 G
## 10 PG-13 PG
## 11 PG-13 PG-13
## 12 PG-13 R
```

```
## 13 R     G
## 14 R     PG
## 15 R     PG-13
## 16 R     R
```

This does "all possible combinations" of one rating with another. We don't actually need all of that; we just need the ones where the first one is (alphabetically) strictly less than the second one. This is because we're never comparing a rating with itself, and each pair of ratings appears twice, once in alphabetical order, and once the other way around. The ones we need are these:

```
crossing(first = the_ratings, second = the_ratings) %>%
    filter(first < second)
```

```
## # A tibble: 6 x 2
##   first second
##   <chr> <chr>
## 1 G     PG
## 2 G     PG-13
## 3 G     R
## 4 PG    PG-13
## 5 PG    R
## 6 PG-13 R
```

A technique thing to note: instead of asking "how do I pick out the distinct pairs of ratings?", I use two simpler tools: first I make all the combinations of pairs of ratings, and then out of those, pick the ones that are alphabetically in ascending order, which we know how to do.

Now we want to call our function `comp2` for each of the things in `first` *and* each of the things in `second`, and make a new column called `pval` that contains exactly that. This (coming fresh from page 332 of the R book, this being the first time I've ever used it) is exactly what the texttt{map2} family of functions does. In our case, `comp2` returns a decimal number, a `dbl`, so `map2_dbl` does it. Thus:

This was a year ago when I first wrote this.

```
crossing(first = the_ratings, second = the_ratings) %>%
    filter(first < second) %>% mutate(pval = map2_dbl(first,
    second, ~comp2(.x, .y, movies)))
```

```
## # A tibble: 6 x 3
##   first second      pval
##   <chr> <chr>      <dbl>
## 1 G     PG     0.00799
## 2 G     PG-13  0.0000590
```

```
## 3 G     R      0.0106
## 4 PG    PG-13  0.0106
## 5 PG    R      0.715
## 6 PG-13 R      0.273
```

The logic of `map2_dbl` is "for each of the things in `first`, and each of the things in `second`, taken in parallel, call the function `comp2` with those two inputs in that order, always with data frame `movies`". The `.x` and `.y` play the role of the `.` that we usually have inside a map, but now we're "mapping" over two things rather than just one, so that they cannot both be called `.`.

One more thing: we're doing 6 tests at once here, so we're giving ourselves 6 chances to reject a null (all medians equal) that might have been true. So the true probability of a type I error is no longer 0.05 but something bigger.

The easiest way around that is to do a so-called Bonferroni adjustment: instead of rejecting if the P-value is less than 0.05, we only reject if it is less than 0.05/6, since we are doing 6 tests. This is a fiddly calculation to do by hand, but it's easy to build in another `mutate`, thus:

In the pairwise median test in *smmr*, I did this backwards: rather than changing the alpha that you compare each P-value with from 0.05 to 0.05/6, I flip it around so that you adjust the P-values by *multiplying* them by 6, and then comparing the adjusted P-values with the usual 0.05. It comes to the same place in the end, except that this way you can get adjusted P-values that are greater than 1, which makes no sense. You read those as being definitely not significant.

```
crossing(first = the_ratings, second = the_ratings) %>%
    filter(first < second) %>% mutate(pval = map2_dbl(first,
    second, ~comp2(.x, .y, movies))) %>% mutate(reject = pval <
    0.05/6)
```

```
## # A tibble: 6 x 4
##   first second      pval reject
##   <chr> <chr>      <dbl> <lgl>
## 1 G     PG     0.00799   TRUE
## 2 G     PG-13  0.0000590 TRUE
## 3 G     R      0.0106    FALSE
## 4 PG    PG-13  0.0106    FALSE
## 5 PG    R      0.715     FALSE
## 6 PG-13 R      0.273     FALSE
```

And not a loop in sight.

This is how I coded it in `pairwise_median_test`. If you want to check it, it's on Github: link. The function `median_test_pair` is the same as `comp2` above.

So the only significant differences are now G compared to PG and PG-13. There is not a significant difference in median movie length between G and R, though it is a close call. We thought the PG-13 movies might have a significantly different median from other rating groups beyond G, but they turn out not to have. (The third and fourth comparisons would have been significant had we not

made the Bonferroni adjustment to compensate for doing six tests at once; with that adjustment, we only reject if the P-value is less than $0.05/6 = 0.0083$, and so 0.0106 is not quite small enough to reject with.)

Listing the rating groups sorted by median would give you an idea of how far different the medians have to be to be significantly different:

```
medians = movies %>% group_by(rating) %>% summarize(med = median(length)) %>%
    arrange(desc(med))
medians
```

```
## # A tibble: 4 x 2
##   rating   med
##   <chr>  <int>
## 1 PG-13    117
## 2 R        103
## 3 PG       100
## 4 G         82
```

Something rather interesting has happened: even though the comparison of G and PG (18 apart) is significant, the comparison of G and R (21 apart) is not significant. This seems very odd, but it happens because the Mood median test is not actually literally comparing the sample medians, but only assessing the splits of values above and below the median of the combined sample. A subtlety, rather than an error, I'd say.

Here's something extremely flashy to finish with:

```
crossing(first = the_ratings, second = the_ratings) %>%
    filter(first < second) %>% mutate(pval = map2_dbl(first,
    second, ~comp2(.x, .y, movies))) %>% mutate(reject = pval <
    0.05/6) %>% left_join(medians, by = c(first = "rating")) %>%
    left_join(medians, by = c(second = "rating"))
```

```
## # A tibble: 6 x 6
##   first second      pval reject med.x med.y
##   <chr> <chr>      <dbl> <lgl>  <int> <int>
## 1 G     PG     0.00799   TRUE      82   100
## 2 G     PG-13  0.0000590 TRUE      82   117
## 3 G     R      0.0106    FALSE     82   103
## 4 PG    PG-13  0.0106    FALSE    100   117
## 5 PG    R      0.715     FALSE    100   103
## 6 PG-13 R      0.273     FALSE    117   103
```

The additional two lines look up the medians of the rating groups in `first`, then `second`, so that you can see the actual medians of the

groups being compared each time. You see that medians different by 30 are definitely different, ones differing by 15 or less are definitely not different, and ones differing by about 20 could go either way.

I think that's *quite* enough of that.

11.4 *Atomic weight of carbon*

The atomic weight of the chemical element carbon is 12. Two methods of measuring the atomic weight of samples of carbon were compared. The results are shown in link. The methods are labelled 1 and 2. The first task is to find out whether the two methods have different "typical" measures (mean or median, as appropriate) of the atomic weight of carbon.

For this question, compose a report in R Markdown. (R Markdown is what you use in an R Notebook, but you can also have a separate R Markdown document from which you can produce HTML, Word etc. output.) See part (a) for how to get this started.

Your report should read like an actual report, not just the answers to some questions that I set you. To help with that, write some text that links the parts of the report together smoothly, so that it reads as a coherent whole. The grader had 3 discretionary marks to award for the overall quality of your writing. The scale for this was:

- 3 points: excellent writing. The report flows smoothly, is easy to read, and contains everything it should (and nothing it shouldn't).
- 2 points: satisfactory writing. Not the easiest to read, but says what it should, and it looks at least somewhat like a report rather than a string of answers to questions.
- 1 point: writing that is hard to read or to understand. If you get this (or 0), you should consider what you need to do to improve when you write your project.
- 0 points: you answered the questions, but you did almost nothing to make it read like a report.

(a) Create a new R Markdown document. To do this, in R Studio, select File, New File, R Markdown. Type the report title and your name in the boxes, and leave the output on the default HTML. Click OK.

Solution

You'll see the title and your name in a section at the top of the document, and below that you'll see a template document, as you would for an R Notebook. The difference is that where you are used

to seeing Preview, it now says "knit", but this has the same effect of producing the formatted version of your report.

(b) Write an introduction that explains the purpose of this study and the data collected in your own words.

Solution
Something like this:

> This study is intended to compare two different methods (labelled 1 and 2) for measuring the atomic weight of carbon (which is known in actual fact to be 12). Fifteen samples of carbon were used; ten of these were assessed using method 1 and the remaining five using method 2. The primary interest in this particular study is to see whether there is a difference in the mean or median atomic weight as measured by the two methods.

Before that, start a new section like this: `## Introduction`. Also, get used to expressing your understanding in your words, not mine. (Using my words, in this course, is likely to be worth very little.)

(c) Begin an appropriately-titled new section in your report, read the data into R and display the results.

Solution
Values separated by spaces:

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

```
## Parsed with column specification:
## cols(
##   method = col_integer(),
##   weight = col_double()
## )
```

```
carbon
```

```
## # A tibble: 15 x 2
##    method weight
##     <int>  <dbl>
##  1      1   12.0
##  2      1   12.0
##  3      1   12.0
##  4      1   12.0
##  5      1   12.0
##  6      1   12.0
##  7      1   12.0
```

```
##  8       1   12.0
##  9       1   12.0
## 10       1   12.0
## 11       2   12.0
## 12       2   12.0
## 13       2   12.0
## 14       2   12.0
## 15       2   12.0
```

I would expect you to include, without being told to include it, some text in your report indicating that you have sensible data: two methods labelled 1 and 2 as promised, and a bunch of atomic weights close to the nominal figure of 12.

It's probably better in a report to use language a bit more formal than *a bunch*. Something like *a number* would be better.

(d) Make an appropriate plot to compare the measurements obtained by the two methods. You might need to do something about the two methods being given as numbers even though they are really only identifiers. (If you do, your report ought to say what you did and why.)

Solution

The appropriate plot, with a categorical method and quantitative weight, is something like a boxplot. If you're not careful, `method` will get treated as a quantitative variable, which you don't want; the easiest way around that, for a boxplot at least, is to turn it into a factor like this:

```
ggplot(carbon, aes(x = factor(method), y = weight)) +
    geom_boxplot()
```

If you insist, you could do a faceted histogram (above and below, for preference):

```
ggplot(carbon, aes(x = weight)) + geom_histogram(bins = 5) +
    facet_wrap(~method, ncol = 1)
```

There are really not enough data values for a histogram to be of much help, so I don't like this as much.

If you are thinking ahead (we are going to be doing a t-test), then you'll realize that normality is the kind of thing we're looking for, in which case normal quantile plots would be the thing. However, we might have to be rather forgiving for method 2 since there are only 5 observations:

```
ggplot(carbon, aes(sample = weight)) + stat_qq() +
    stat_qq_line() + facet_wrap(~method)
```

I don't mind these coming out side by side, though I would rather have them squarer.

I would say, boxplots are the best, normal quantile plots are also acceptable, but expect to lose something for histograms because they

offer only a rather crude comparison in this case.

(e) Comment briefly on what you see in your plot.

Solution

In boxplots, if that's what you drew, there are several things that deserve comment: the medians, the spreads and the shapes. The median for method 1 is a little bit lower than for method 2 (the means are probably more different, given the shapes of the boxes). The spread for method 2 is a lot bigger. (Looking forward, that suggests a Welch-Satterthwaite rather than a pooled test.) As for shape, the method 2 measurements seem more or less symmetric (the whiskers are equal anyway, even if the position of the median in the box isn't), but the method 1 measurements have a low outlier. The histograms are hard to compare. Try to say something about centre and spread and shape. I think the method 2 histogram has a slightly higher centre and definitely bigger spread. On my histogram for method 1, the distribution looks skewed left. If you did normal quantile plots, say something sensible about normality for each of the two methods. For method 1, I would say the low value is an outlier and the rest of the values look pretty straight. Up to you whether you think there is a curve on the plot (which would indicate skewness, but then that highest value is too high: it would be bunched up with the other values below 12.01 if there were really skewness). For method 2, it's really hard to say anything since there are only five values. Given where the line goes, there isn't much you can say to doubt normality. Perhaps the best you can say here is that in a sample of size 5, it's difficult to assess normality at all.

(f) Carry out the most appropriate t-test. (You might like to begin another new section in your report here.)

Solution

This would be the Welch-Satterthwaite version of the two-sample t-test, since the two groups do appear to have different spreads:

```
t.test(weight ~ method, data = carbon)
```

```
##
##  Welch Two Sample t-test
##
## data:  weight by method
## t = -1.817, df = 5.4808, p-value =
## 0.1238
## alternative hypothesis: true difference in means is not equal to 0
## 95 percent confidence interval:
```

```
##  -0.027777288  0.004417288
## sample estimates:
## mean in group 1 mean in group 2
##        12.00260        12.01428
```

Imagining that this is a report that would go to your boss, you ought to defend your choice of the Welch-Satterthwaite test (as I did above), and not just do the default *t*-test without comment.

If, in your discussion above, you thought the spreads were equal enough, then you should do the pooled *t*-test here, which goes like this:

```
t.test(weight ~ method, data = carbon, var.equal = T)
```

```
##
##  Two Sample t-test
##
## data:  weight by method
## t = -2.1616, df = 13, p-value = 0.04989
## alternative hypothesis: true difference in means is not equal to 0
## 95 percent confidence interval:
##  -2.335341e-02 -6.588810e-06
## sample estimates:
## mean in group 1 mean in group 2
##        12.00260        12.01428
```

The point here is that you should do the right test based on your conclusion. Being consistent is the most important thing. (In this case, note that the P-values are very different. We'll get to that shortly.)

If we were doing this in SAS, as we see later, we'd get a test at the bottom of the output that compares the two variances. I feel that it's just as good to eyeball the spreads and make a call about whether they are "reasonably close". Or even, to always do the Welch-Satterthwaite test on the basis that it is pretty good even if the two populations have the same variance. (If this last point of view is one that you share, you ought to say something about that when you do your *t*-test.)

Extra: I guess this is a good place to say something about tests for comparing variances, given that you might be pondering that. There are several that I can think of, that R can do, of which I mention two.

The first is the *F*-test for variances that you might have learned in B57 (that is the basis for the ANOVA *F*-test):

```
var.test(weight ~ method, data = carbon)
```

```
##
##  F test to compare two variances
##
## data:  weight by method
## F = 0.35768, num df = 9, denom df = 4,
## p-value = 0.1845
## alternative hypothesis: true ratio of variances is not equal to 1
## 95 percent confidence interval:
##  0.04016811 1.68758230
## sample estimates:
## ratio of variances
##          0.3576842
```

This, unfortunately, is rather dependent on the data in the two groups being approximately normal. Since we are talking variances rather than means, there is no Central Limit Theorem to rescue us for large samples (quite aside from the fact that these samples are *not* large). Since the ANOVA F-test is based on the same theory, this is why normality is also more important in ANOVA than it is in a t-test.

The second is Levene's test. This doesn't depend on normality (at least, not nearly so much), so I like it better in general:

```r
library(car)
```

```
## Loading required package: carData

##
## Attaching package: 'car'

## The following object is masked from 'package:dplyr':
##
##     recode

## The following object is masked from 'package:purrr':
##
##     some
```

```r
leveneTest(weight ~ factor(method), data = carbon)
```

```
## Levene's Test for Homogeneity of Variance (center = median)
##       Df F value Pr(>F)
## group  1  0.9887 0.3382
##       13
```

Levene's test takes a different approach: first the absolute differences from the group medians are calculated, and then an ANOVA is

run on the absolute differences. If, say, one of the groups has a larger spread than the other(s), its absolute differences from the median will tend to be bigger. As for what we conclude here, well, neither of the variance tests show any significance at all, so from that point of view there is no evidence against using the pooled t-test. Having said that, the samples are small, and so it would be difficult to *prove* that the two methods have different variance, even if they actually did.

The use of absolute differences, and the median, downplays the influence of outliers. The assumption here is that the absolute differences from the medians are approximately normal, which seems a less big assumption than assuming the actual data are approximately normal.

This is coming back to the *power* of something like Levene's test; the power of any test is not going to be very big if the sample sizes are small.

Things are never as clear-cut as you would like. In the end, it all comes down to making a call and defending it.

(g) Do the most appropriate test you know that does not assume normally-distributed data.

Solution

That would be Mood's median test. Since I didn't say anything about building it yourself, feel free to use `smmr`:

```
library(smmr)
median_test(carbon, weight, method)
```

```
## $table
##      above
## group above below
##     1     3     6
##     2     4     1
##
## $test
##        what      value
## 1 statistic 2.80000000
## 2        df 1.00000000
## 3   P-value 0.09426431
```

As an aside, if you have run into a non-parametric test such as Mann-Whitney or Kruskal-Wallis that applies in this situation, be careful about using it here, because they have additional assumptions that you may not want to trust. Mann-Whitney started life as a test for "equal distributions". This means that the null is equal location *and* equal spread, and if you reject the null, one of those has failed. But here, we suspect that equal spread will fail, so that the Mann-Whitney test may end up rejecting *whether or not* the medians are different, so it won't answer the question you want an answer to. Mood's median test doesn't have that problem; all it's saying if the null is true is that the medians are equal; the spreads could be anything at all.

The test goes back to the 1940s.

The same kind of issues apply to the signed-rank test vs. the sign test. In the case of the signed-rank test, the extra assumption is of a

symmetric distribution — to my mind, if you don't believe normality, you probably don't have much confidence in symmetry either. That's why I like the sign test and Mood's median test: in the situation where you don't want to be dealing with assumptions, these tests don't make you worry about that.

Another comment that you don't need to make is based on the not-quite-significance of the Mood test. The P-value is less than 0.10 but not less than 0.05, so it doesn't quite reach significance by the usual standard. But if you look up at the table, the frequencies seem rather unbalanced: 6 out of the remaining 9 weights in group 1 are below the overall median, but 4 out of 5 weights in group 2 are above. This seems as if it ought to be significant, but bear in mind that the sample sizes are small, and thus Mood's median test needs *very* unbalanced frequencies, which we don't quite have here.

(h) Discuss the results of your tests and what they say about the two methods for measuring the atomic weight of carbon. If it seems appropriate, put the discussion into a section called Conclusions.

Solution

Begin by pulling out the P-values for your preferred test(s) and say what they mean. The P-value for the Welch-Satterthwaite t-test is 0.1238, which indicates no difference in mean atomic weights between the two methods. The Mood median test gives a similarly non-significant 0.0943, indicating no difference in the *median* weights. If you think both tests are plausible, then give both P-values and do a compare-and-contrast with them; if you think that one of the tests is clearly preferable, then say so (and why) and focus on that test's results. If you thought the pooled test was the right one, then you'll have a bit more discussion to do, since its P-value is 0.0499, and at $\alpha = 0.05$ this test disagrees with the others. If you are comparing this test with the Mood test, you ought to make some kind of reasoned recommendation about which test to believe. As ever, be consistent in your reasoning. This dataset, where I found it, was actually being used to illustrate a case where the pooled and the Welch-Satterthwaite tests disagreed. The authors of the original paper that used this dataset (a 1987 paper by Rayner and Best, [1]; the data come from 1924!) point out that the pooled t-test can be especially misleading when the smaller sample is also the one with the larger variance. This is what happened here. In the Rayner and Best paper, the Mood (or the Mann-Whitney) test was not being considered, but I think it's good practice to draw a picture and make a call about which test is appropriate. I loaded package `car` above; I'd better be tidy and unload it before I go on:

```
detach(package:car, unload = T)
```

11.5 *Can caffeine improve your performance on a test?*

Does caffeine help students do better on a certain test? To find out, 36 students were randomly allocated to three groups (12 in each group). Each student received a fixed number of cups of coffee while they were studying, but the students didn't know whether they were receiving all full-strength coffee ("high"), all decaf coffee ("low") or a 50-50 mixture of the two ("moderate"). For each subject, their group was recorded as well as their score on the test. The data are in link, as a `.csv` file.

(a) Read in and examine the data. How are the values laid out?

Solution

`read_csv` because it's a `.csv` file:

```
my_url = "http://www.utsc.utoronto.ca/~butler/d29/caffeine.csv"
caffeine.untidy = read_csv(my_url)
```

```
## Parsed with column specification:
## cols(
##   Sub = col_integer(),
##   High = col_integer(),
##   Moderate = col_integer(),
##   None = col_integer()
## )
```

```
caffeine.untidy
```

```
## # A tibble: 12 x 4
##      Sub  High Moderate  None
##    <int> <int>    <int> <int>
##  1     1    72       68    68
##  2     2    65       80    74
##  3     3    68       64    59
##  4     4    83       65    61
##  5     5    79       69    65
##  6     6    92       79    72
##  7     7    69       80    80
##  8     8    74       63    58
##  9     9    78       69    65
## 10    10    83       70    60
## 11    11    88       83    78
## 12    12    71       75    75
```

The first column is the number of the subject (actually within each group, since each student only tried one amount of caffeine). Then

follow the test scores for the students in each group, one group per column.

I gave the data frame a kind of dumb name, since (looking ahead) I could see that I would need a less-dumb name for the tidied-up data, and it seemed sensible to keep `caffeine` for that.

(b) Explain briefly how the data are not "tidy".

Solution

The last three columns are all scores on the test: that is, they all measure the same thing, so they should all be in the same column. Or, there should be a column of scores, and a separate column naming the groups. Or, there were 36 observations in the data, so there should be 36 rows. You always have a variety of ways to answer these, any of which will do.

(c) Use a suitable tool from the `tidyverse` to create one column of test scores and and one column of group labels. Call your column of group labels `amount`. Is it a `factor`?

Solution

We are combining several columns into one, so this is `gather`:

```
caffeine = caffeine.untidy %>% gather(amount,
    score, c(High:None))
```

I didn't ask you to list the resulting data frame, but it is smart to at least look for yourself, to make sure `gather` has done what you expected.

```
caffeine
```

```
## # A tibble: 36 x 3
##      Sub amount score
##    <int> <chr>  <int>
##  1     1 High      72
##  2     2 High      65
##  3     3 High      68
##  4     4 High      83
##  5     5 High      79
##  6     6 High      92
##  7     7 High      69
##  8     8 High      74
##  9     9 High      78
## 10    10 High      83
## # ... with 26 more rows
```

A column of amounts of caffeine, and a column of test scores. This is what we expected. There should be 12 each of the `amounts`, which you can check if you like:

```
caffeine %>% count(amount)
```

```
## # A tibble: 3 x 2
##   amount       n
##   <chr>    <int>
## 1 High        12
## 2 Moderate    12
## 3 None        12
```

Indeed.

Note that `amount` is text, not a factor. Does this matter? We'll see.

This is entirely the kind of situation where you need `gather`, so get used to seeing where it will be useful.

(d) Obtain side-by-side boxplots of test scores by amount of caffeine.

Solution

```
ggplot(caffeine, aes(x = amount, y = score)) +
    geom_boxplot()
```

(e) Does caffeine amount seem to have an effect? If so, what kind of effect?

Solution

On average, exam scores seem to be higher when the amount of caffeine is higher (with the effect being particularly pronounced for High caffeine). If you want to, you can also say the the effect of

caffeine seems to be small, relative to the amount of variability there is (there is a lot). The point is that you say *something* supported by the boxplot.

(f) Run a suitable analysis of variance to determine whether the mean test score is equal or unequal for the three groups. What do you conclude?

Solution
Something like this:

```
caff.1 = aov(score ~ amount, data = caffeine)
summary(caff.1)
```

```
##             Df Sum Sq Mean Sq F value Pr(>F)
## amount       2  477.7  238.86   3.986 0.0281
## Residuals   33 1977.5   59.92
##
## amount      *
## Residuals
## ---
## Signif. codes:
##   0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1  ' ' 1
```

The P-value on the F-test is less than 0.05, so we reject the null hypothesis (which says that all the groups have equal means) in favour of the alternative: the group means are not all the same (one or more of them is different from the others).

Notice that the boxplot and the `aov` are quite happy for `amount` to be text rather than a factor (they actually do want a factor, but if the input is text, they'll create one).

(g) Why is it a good idea to run Tukey's method here?

Solution
The analysis of variance F-test is significant, so that the groups are not all the same. Tukey's method will tell us which group(s) differ(s) from the others. There are three groups, so there are differences to find that we don't know about yet.

(h) Run Tukey's method. What do you conclude?

Solution
This kind of thing:

```
caff.3 = TukeyHSD(caff.1)
caff.3
```

```
##   Tukey multiple comparisons of means
##     95% family-wise confidence level
##
## Fit: aov(formula = score ~ amount, data = caffeine)
##
## $amount
##                    diff       lwr       upr
## Moderate-High -4.750000 -12.50468  3.004679
## None-High     -8.916667 -16.67135 -1.161987
## None-Moderate -4.166667 -11.92135  3.588013
##                   p adj
## Moderate-High 0.3025693
## None-High     0.0213422
## None-Moderate 0.3952176
```

The high-caffeine group definitely has a higher mean test score than the no-caffeine group. (The Moderate group is not significantly different from either of the other groups.) Both the comparisons involving Moderate could go either way (the interval for the difference in means includes zero). The None-High comparison, however, is away from zero, so this is the significant one. As is usual, we are pretty sure that the difference in means (this way around) is negative, but we are not at all clear about how big it is, because the confidence interval is rather long.

We'd need a lot more students to make it narrower, but this is not surprising since students vary in a lot of other ways that were not measured here.

12
Tidying and selecting data

```r
library(tidyverse)
```

```
## -- Attaching packages ---- tidyverse 1.2.1 --
```

```
## v ggplot2 3.1.0     v purrr   0.2.5
## v tibble  1.4.2     v dplyr   0.7.8
## v tidyr   0.8.1     v stringr 1.3.1
## v readr   1.1.1     v forcats 0.3.0
```

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

12.1 *Tidying the Jays data*

This question is about the Blue Jays data set (that I used in class).

(a) The Blue Jays baseball data set is at link. Read it into R. Check that you have 25 rows and a bunch of variables.

Solution

Save the URL into a variable and then read from the URL, using `read_csv` because it's a `.csv` file:

```r
myurl = "http://www.utsc.utoronto.ca/~butler/c32/jays15-home.csv"
jays = read_csv(myurl)
```

```
## Parsed with column specification:
## cols(
##   .default = col_character(),
##   row = col_integer(),
##   game = col_integer(),
##   runs = col_integer(),
##   Oppruns = col_integer(),
```

```
##   innings = col_integer(),
##   position = col_integer(),
##   `game time` = col_time(format = ""),
##   attendance = col_integer()
## )

## See spec(...) for full column specifications.
```

```
jays
```

```
## # A tibble: 25 x 21
##      row  game date  box   team  venue opp
##    <int> <int> <chr> <chr> <chr> <chr> <chr>
##  1    82     7 Mond~ boxs~ TOR   <NA>  TBR
##  2    83     8 Tues~ boxs~ TOR   <NA>  TBR
##  3    84     9 Wedn~ boxs~ TOR   <NA>  TBR
##  4    85    10 Thur~ boxs~ TOR   <NA>  TBR
##  5    86    11 Frid~ boxs~ TOR   <NA>  ATL
##  6    87    12 Satu~ boxs~ TOR   <NA>  ATL
##  7    88    13 Sund~ boxs~ TOR   <NA>  ATL
##  8    89    14 Tues~ boxs~ TOR   <NA>  BAL
##  9    90    15 Wedn~ boxs~ TOR   <NA>  BAL
## 10    91    16 Thur~ boxs~ TOR   <NA>  BAL
## # ... with 15 more rows, and 14 more
## #   variables: result <chr>, runs <int>,
## #   Oppruns <int>, innings <int>, wl <chr>,
## #   position <int>, gb <chr>, winner <chr>,
## #   loser <chr>, save <chr>, `game
## #   time` <time>, Daynight <chr>,
## #   attendance <int>, streak <chr>
```

If you must, copy and paste the spreadsheet into R Studio, and read it in with `read_delim` (or possibly `read_tsv`), but this runs the risk of being defeated by spreadsheet cells that contain spaces. I don't think there are any here, but you might run into a pitcher whose name has more than one word, like (Andy) Van Hekken, who is in the Seattle Mariners farm system.

I found this by googling, after I had scrolled past all the pages of articles about the baseball pitcher who *lives* in a van.

Anyway, 25 rows and 21 columns. As usual, it's a tibble, so you see 10 rows and as many columns as will fit. This is often enough to see whether we have the right thing (as we appear to have, here). You can run through all the columns and check that they're the right kind of thing; most of them are text with a few numbers and one `time`, which is `game time`, the length of the game in hours and minutes, which is turned into an R `time` in hours, minutes and seconds.

With all those columns, `read_csv` doesn't tell you what column specification it inferred for all of them, but you can type

```r
spec(jays)
```

```
## cols(
##   row = col_integer(),
##   game = col_integer(),
##   date = col_character(),
##   box = col_character(),
##   team = col_character(),
##   venue = col_character(),
##   opp = col_character(),
##   result = col_character(),
##   runs = col_integer(),
##   Oppruns = col_integer(),
##   innings = col_integer(),
##   wl = col_character(),
##   position = col_integer(),
##   gb = col_character(),
##   winner = col_character(),
##   loser = col_character(),
##   save = col_character(),
##   `game time` = col_time(format = ""),
##   Daynight = col_character(),
##   attendance = col_integer(),
##   streak = col_character()
## )
```

to find it all out.

(b) Pick out only the games that were against the New York Yankees (the variable `opp` is equal to `NYY`). Investigate all the columns. What do you notice about these games?

Solution
I get to do this:

```r
jays %>% filter(opp == "NYY") %>% print(width = Inf)
```

```
## # A tibble: 3 x 21
##     row  game date             box      team
##   <int> <int> <chr>            <chr>    <chr>
## 1    92    27 Monday, May 4    boxscore TOR
## 2    93    28 Tuesday, May 5   boxscore TOR
## 3    94    29 Wednesday, May 6 boxscore TOR
##   venue opp   result  runs Oppruns innings
##   <chr> <chr> <chr>  <int>   <int>   <int>
## 1 <NA>  NYY   W          3       1      NA
```

```
## 2 <NA>  NYY   L          3       6      NA
## 3 <NA>  NYY   W          5       1      NA
##   wl    position gb    winner  loser
##   <chr>    <int> <chr> <chr>   <chr>
## 1 13-14        4 3.5   Dickey  Martin
## 2 13-15        5 4.5   Pineda  Estrada
## 3 14-15        3 3.5   Buehrle Sabathia
##   save   `game time` Daynight attendance
##   <chr>  <time>      <chr>         <int>
## 1 Cecil  02:18       N             19217
## 2 Miller 02:54       N             21519
## 3 <NA>   02:30       N             21312
##   streak
##   <chr>
## 1 +
## 2 -
## 3 +
```

but you will probably need to click the little right-arrow at the top to see more columns.

What I notice is that these games are all on consecutive nights (against the same team). This is quite common, and goes back to the far-off days when teams travelled by train: teams play several games on one visit, rather than coming back many times. You might have noticed something else; that's fine for this. For example, "each of the games lasted less than three hours", or "the attendances were all small" (since we looked at all the attendances in class). I just want you to notice something meaningful that seems to be interesting about these games.

Hockey is similar: teams go on road trips, playing several different teams before returning home. Hockey teams, though, tend to play each team only once on a road trip: for example, a west coast team like the Canucks might play a game in each of Toronto, Montreal, Boston and New York on a road trip. Well, maybe three games in the New York area: one each against the Rangers, Islanders and Devils.

You could also print all the columns in two or more goes, using `select`, for example:

```
jays %>% filter(opp == "NYY") %>% select(row:innings) %>%
    print(width = Inf)
```

```
## # A tibble: 3 x 11
##     row  game date             box      team
##   <int> <int> <chr>            <chr>    <chr>
## 1    92    27 Monday, May 4    boxscore TOR
## 2    93    28 Tuesday, May 5   boxscore TOR
## 3    94    29 Wednesday, May 6 boxscore TOR
##   venue opp   result  runs Oppruns innings
##   <chr> <chr> <chr>  <int>   <int>   <int>
## 1 <NA>  NYY   W          3       1      NA
## 2 <NA>  NYY   L          3       6      NA
## 3 <NA>  NYY   W          5       1      NA
```

```
jays %>% filter(opp == "NYY") %>% select(wl:streak) %>%
    print(width = Inf)
```

```
## # A tibble: 3 x 10
##   wl    position gb    winner  loser
##   <chr>    <int> <chr> <chr>   <chr>
## 1 13-14        4 3.5   Dickey  Martin
## 2 13-15        5 4.5   Pineda  Estrada
## 3 14-15        3 3.5   Buehrle Sabathia
##   save   `game time` Daynight attendance
##   <chr>  <time>      <chr>         <int>
## 1 Cecil  02:18       N             19217
## 2 Miller 02:54       N             21519
## 3 <NA>   02:30       N             21312
##   streak
##   <chr>
## 1 +
## 2 -
## 3 +
```

(c) From the whole data frame, pick out only the games where the attendance was more than 30,000, showing only the columns `attendance` and `Daynight`. How many of them are there (just count them)? How many are day games and how many night games (just count those too)?

Solution

Two steps, since we selecting rows *and* columns.

```
jays %>% filter(attendance > 30000) %>% select(c(attendance,
    Daynight))
```

```
## # A tibble: 8 x 2
##   attendance Daynight
##        <int> <chr>
## 1      48414 N
## 2      34743 D
## 3      44794 D
## 4      30430 N
## 5      42917 D
## 6      42419 D
## 7      33086 D
## 8      37929 D
```

Or this way, since we are selecting *consecutive* columns:

```
jays %>% filter(attendance > 30000) %>% select(c(Daynight:attendance))
```

```
## # A tibble: 8 x 2
##   Daynight attendance
##   <chr>         <int>
## 1 N             48414
## 2 D             34743
## 3 D             44794
## 4 N             30430
## 5 D             42917
## 6 D             42419
## 7 D             33086
## 8 D             37929
```

There are eight games selected (see the eight rows in the result). Only two of them are night games, while the other six are day (weekend) games.

If you wanted to, you could automate the counting, like this:

```
jays %>% filter(attendance > 30000) %>% count(Daynight)
```

```
## # A tibble: 2 x 2
##   Daynight     n
##   <chr>    <int>
## 1 D            6
## 2 N            2
```

Six day games and two night games.

(d) Display the mean and standard deviation of attendances at all day and night games.

Solution

Two steps: the grouping according to what I want to group by, then summarizing according to what I want to summarize by. Since I am summarizing, only the summaries find their way into the final data frame, so I don't need to "select out" the other variables:

```
jays %>% group_by(Daynight) %>% summarize(mean.att = mean(attendance),
    sd.att = sd(attendance))
```

```
## # A tibble: 2 x 3
##   Daynight mean.att sd.att
##   <chr>       <dbl>  <dbl>
## 1 D          37885.  5775.
## 2 N          20087.  8084.
```

The mean attendances are about 38 thousand and about 20 thousand. Note that the night games have much the larger SD, possibly because of the large outlier night attendance (opening night). Which we can also investigate.

```
jays %>% group_by(Daynight) %>% summarize(median.att = median(attendance),
    iqr.att = IQR(attendance))
```

```
## # A tibble: 2 x 3
##   Daynight median.att iqr.att
##   <chr>         <dbl>   <dbl>
## 1 D            37929    8754.
## 2 N            17928.   6005.
```

This time, the night attendances have a *smaller* spread and a noticeably smaller median (compared to the mean), so it must have been the outlier that made the difference. There was another high value that R marked as an outlier:

```
ggplot(jays, aes(x = Daynight, y = attendance)) +
    geom_boxplot()
```

So when you take away those unusual values, the night game attendances are indeed less variable.

The right test, as you might have guessed, for comparing the medians of these non-normal data, is Mood's median test:

```
library(smmr)
median_test(jays, attendance, Daynight)
```

```
## $table
##      above
## group above below
##     D     7     0
##     N     5    12
##
## $test
```

```
##        what       value
## 1 statistic 9.882352941
## 2        df 1.000000000
## 3   P-value 0.001668714
```

There was one attendance exactly equal to the overall median (as you would expect: with an odd number of data values, the median is one of the data values). `smmr` removed it; if you did the test by hand, what happens to it depends on whether you counted aboves or belows, and this will have a small effect on the P-value, though not on the conclusion.

The overall median attendance was 21,000, and *none* of the day games had attendance less than that. With the small frequencies, the accuracy of the P-value is a bit questionable, but taking it at face value, there *is* a significant difference between median attendances at day and night games.

If you do this by hand, you'll get a warning about the chi-squared approximation being inaccurate. This is because of the small frequencies, and *not* because of the outliers. Those are not damaging the test at all.

(e) Make normal quantile plots of the day attendances and the night attendances, separately. Do you see any evidence of non-normality? (You would expect to on the night attendances because of the big opening-night value.)

Solution

The best way to do this is facetted normal quantile plots. Remember that the facetting part goes right at the end:

```
ggplot(jays, aes(sample = attendance)) + stat_qq() +
    stat_qq_line() + facet_wrap(~Daynight, ncol = 1)
```

The day attendances are pretty normal, though it is hard to be sure with only 7 of them.

The night attendances are not normal. The lone point top right is the outlier. On top of that, the lowest attendances are not quite low

enough and the second-highest attendance is a bit too high, so there is a bit of evidence of right-skewness as well as just the one outlier.

If you leave out the `ncol=1`, you'll get the two normal quantile plots side by side (which means that each one is tall and skinny, and thus hard to read). The `ncol=1` displays all the facets in *one* column, and though it would be nice to have the graphs be about square, landscape mode is easier to read than portrait mode.

One of the reasons for skewness is often a limit on the values of the variable. The Rogers Centre has a capacity around 55,000. The day game attendances don't get especially close to that, which suggests that everyone who wants to go to the game can get a ticket. In that sort of situation, you'd expect attendances to vary around a "typical" value, with a random deviation that depends on things like the weather and the opposing team, which is the typical situation in which you get bell-shaped data. (If the Jays often sold out their stadium for day games, you'd see a lot of attendances close to the capacity, with a few lower: ie., a left skew.)

As for the night games, well, there seems to be a minimum attendance that the Blue Jays get, somewhere around 15,000: no matter who they're playing or what the weather's like, this many people will show up (season-ticket holders, for example). On special occasions, such as opening night, the attendance will be much bigger, which points to a *right* skew.

12.2 *Baseball and softball spaghetti*

On a previous assignment, we found that students could throw a baseball further than they could throw a softball. In this question, we will make a graph called a "spaghetti plot" to illustrate this graphically. (The issue previously was that the data were matched pairs: the same students threw both balls.)

This seems to work most naturally by building a pipe, a line or two at a time. See if you can do it that way. (If you can't, use lots of temporary data frames, one to hold the result of each part.)

(a) Read in the data again from link. The variables had no names, so supply some, as you did before.

Solution

Literal copy and paste:

```
myurl = "http://www.utsc.utoronto.ca/~butler/c32/throw.txt"
throws = read_delim(myurl, " ", col_names = c("student",
    "baseball", "softball"))
```

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

```
## cols(
##   student = col_integer(),
##   baseball = col_integer(),
##   softball = col_integer()
## )
```

```
throws
```

```
## # A tibble: 24 x 3
##    student baseball softball
##      <int>    <int>    <int>
##  1       1       65       57
##  2       2       90       58
##  3       3       75       66
##  4       4       73       61
##  5       5       79       65
##  6       6       68       56
##  7       7       58       53
##  8       8       41       41
##  9       9       56       44
## 10      10       70       65
## # ... with 14 more rows
```

(b) Create a new column that is the students turned into a `factor`, adding it to your data frame.

Solution

Feed `student` into `factor`, creating a new column with `mutate`:

```
throws %>% mutate(fs = factor(student))
```

```
## # A tibble: 24 x 4
##    student baseball softball fs
##      <int>    <int>    <int> <fct>
##  1       1       65       57 1
##  2       2       90       58 2
##  3       3       75       66 3
##  4       4       73       61 4
##  5       5       79       65 5
##  6       6       68       56 6
##  7       7       58       53 7
##  8       8       41       41 8
##  9       9       56       44 9
## 10      10       70       65 10
## # ... with 14 more rows
```

This doesn't look any different from the original student numbers, but note the variable type at the top of the column.

(c) Gather together all the throwing distances into one column, making a second column that says which ball was thrown.

Solution

Literally `gather` (from `tidyr`):

```
throws %>% mutate(fs = factor(student)) %>% gather(ball,
    distance, baseball:softball)
```

```
## # A tibble: 48 x 4
##    student fs    ball     distance
##      <int> <fct> <chr>       <int>
##  1       1 1     baseball       65
##  2       2 2     baseball       90
##  3       3 3     baseball       75
##  4       4 4     baseball       73
##  5       5 5     baseball       79
##  6       6 6     baseball       68
##  7       7 7     baseball       58
##  8       8 8     baseball       41
##  9       9 9     baseball       56
## 10      10 10    baseball       70
## # ... with 38 more rows
```

Two columns to include, consecutive ones, or two to omit, the first and last. So I think it's easier to name the ones you want to include.

If you want to show off a little, you can use a select-helper, noting that the columns you want to gather up all end in "ball":

```
throws %>% mutate(fs = factor(student)) %>% gather(ball,
    distance, ends_with("ball"))
```

```
## # A tibble: 48 x 4
##    student fs    ball     distance
##      <int> <fct> <chr>       <int>
##  1       1 1     baseball       65
##  2       2 2     baseball       90
##  3       3 3     baseball       75
##  4       4 4     baseball       73
##  5       5 5     baseball       79
##  6       6 6     baseball       68
##  7       7 7     baseball       58
##  8       8 8     baseball       41
##  9       9 9     baseball       56
## 10      10 10    baseball       70
## # ... with 38 more rows
```

with the same result. Use whichever you like.

(d) Using your new data frame, make a "scatterplot" of throwing distance against type of ball.

Solution
The obvious thing:

```
throws %>% mutate(fs = factor(student)) %>% gather(ball,
    distance, baseball:softball) %>% ggplot(aes(x = ball,
    y = distance)) + geom_point()
```

What this plot is missing is an indication of which student threw which ball. As it stands now, it could be an inferior version of a boxplot of distances thrown for each ball (which would imply that they are two independent sets of students, something that is not true).

(e) Add two things to your plot: something that will distinguish the students by colour (this works best if the thing distinguished by colour is a factor), and something that will join the two points for the same student by a line.

You can try it without. See below.

Solution
A `colour` and a `group` in the `aes`, and a `geom_line`:

```
throws %>% mutate(fs = factor(student)) %>% gather(ball,
    distance, baseball:softball) %>% ggplot(aes(x = ball,
    y = distance, group = fs, colour = fs)) +
    geom_point() + geom_line()
```

You can see what happens if you use the student as a number:

```
throws %>% mutate(fs = factor(student)) %>% gather(ball,
    distance, baseball:softball) %>% ggplot(aes(x = ball,
    y = distance, group = student, colour = student)) +
    geom_point() + geom_line()
```

Now the student numbers are distinguished as a shade of blue (on an implied continuous scale: even a nonsensical fractional student number like 17.5 would be a shade of blue). This is not actually so bad here, because all we are trying to do is to distinguish the students sufficiently from each other so that we can see where the spaghetti strands go. But I like the multi-coloured one better.

(f) The legend is not very informative. Remove it from the plot, using `guides`.

Solution

You won't have seen this before. Here's what to do: Find what's at the top of the legend that you want to remove. Here that is `fs`. Find where `fs` appears in your `aes`. It actually appears in two places: in `group` and `colour`. I think the legend we want to get rid of is actually the `colour` one, so we do this:

```
throws %>% mutate(fs = factor(student)) %>% gather(ball,
    distance, baseball:softball) %>% ggplot(aes(x = ball,
    y = distance, group = fs, colour = fs)) +
    geom_point() + geom_line() + guides(colour = F)
```

That seems to have done it.

(g) What do you see on the final spaghetti plot? What does that tell you about the relative distances a student can throw a baseball vs. a softball? Explain briefly, blah blah blah.

Solution

Most of the spaghetti strands go downhill from baseball to softball, or at least very few of them go uphill. That tells us that most students can throw a baseball further than a softball. That was the same impression that the matched-pairs t-test gave us. But the spaghetti plot tells us something else. If you look carefully, you see that most of the big drops are for students who could throw a baseball a long way. These students also threw a softball further than the other students, but not by as much. Most of the spaghetti strands in the bottom half of the plot go more or less straight across. This indicates that students who cannot throw a baseball very far will throw a softball about the same distance as they threw the baseball. There is an argument you could make here that the difference between distances thrown is a *proportional* one, something like "a student typically throws a baseball 20% further than a softball". That could be

assessed by comparing not the distances themselves, but the logs of the distances: in other words, making a log transformation of all the distances. (Distances have a lower limit of zero, so you might expect observed distances to be skewed to the right, which is another argument for making some kind of transformation.)

12.3 *Ethanol and sleep time in rats*

A biologist wished to study the effects of ethanol on sleep time in rats. A sample of 20 rats (all the same age) was selected, and each rat was given an injection having a particular concentration (0, 1, 2 or 4 grams per kilogram of body weight) of ethanol. These are labelled `e0, e1, e2, e4`. The "0" treatment was a control group. The rapid eye movement (REM) sleep time was then recorded for each rat. The data are in link.

(a) Read the data in from the file. Check that you have four rows of observations and five columns of sleep times.

Solution
Separated by single spaces:

```
sleep1 = read_delim("ratsleep.txt", " ")
```

```
## Parsed with column specification:
## cols(
##   treatment = col_character(),
##   obs1 = col_double(),
##   obs2 = col_double(),
##   obs3 = col_double(),
##   obs4 = col_double(),
##   obs5 = col_double()
## )
```

```
sleep1
```

```
## # A tibble: 4 x 6
##   treatment  obs1  obs2  obs3  obs4  obs5
##   <chr>     <dbl> <dbl> <dbl> <dbl> <dbl>
## 1 e0         88.6  73.2  91.4  68    75.2
## 2 e1         63    53.9  69.2  50.1  71.5
## 3 e2         44.9  59.5  40.2  56.3  38.7
## 4 e4         31    39.6  45.3  25.2  22.7
```

There are six columns, but one of them labels the groups, and there are correctly five columns of sleep times.

I used a "temporary" name for my data frame, because I'm going to be doing some processing on it in a minute, and I want to reserve the name `sleep` for my processed data frame.

(b) Unfortunately, the data are in the wrong format. All the sleep times for each treatment group are on one row, and we should have *one* column containing *all* the sleep times, and the corresponding row should show which treatment group that sleep time came from. If you prefer to skip this part: read in the data from link, and proceed to the boxplots in (c). The `tidyr` function `gather` turns wide format (which we have) into long format (which we want). `gather` needs four things fed into it: a data frame, what makes the columns different, what makes them the same, and finally which columns are to be `gather`ed together (combined into one column), the first one, a colon, and the last one. Save the result of `gather` into a data frame, and look at it. Do you have 20 rows of not-very-many variables?

Solution

What makes the columns `obs1` through `obs5` different is that they are different observation numbers ("replicates", in the jargon). I'll call that `rep`. What makes them the same is that they are all sleep times. Columns `obs1` through `obs5` are the ones we want to combine, thus. Here is where I use the name `sleep`: I save the result of the `gather` into a data frame `sleep`. Note that I also used the brackets-around-the-outside to display what I had, so that I didn't have to do a separate display. This is a handy way of saving *and* displaying in one shot:

```
(sleep <- sleep1 %>% gather(rep, sleeptime, obs1:obs5))
```

```
## # A tibble: 20 x 3
##    treatment rep   sleeptime
##    <chr>     <chr>     <dbl>
##  1 e0        obs1       88.6
##  2 e1        obs1       63
##  3 e2        obs1       44.9
##  4 e4        obs1       31
##  5 e0        obs2       73.2
##  6 e1        obs2       53.9
##  7 e2        obs2       59.5
##  8 e4        obs2       39.6
##  9 e0        obs3       91.4
## 10 e1        obs3       69.2
## 11 e2        obs3       40.2
## 12 e4        obs3       45.3
```

```
## 13 e0        obs4       68
## 14 e1        obs4       50.1
## 15 e2        obs4       56.3
## 16 e4        obs4       25.2
## 17 e0        obs5       75.2
## 18 e1        obs5       71.5
## 19 e2        obs5       38.7
## 20 e4        obs5       22.7
```

We have 20 rows of 3 columns. I got all the rows, but you will probably get an output with ten rows as usual, and will need to click Next to see the last ten rows. The initial display will say how many rows (20) and columns (3) you have.

The column `rep` is not very interesting: it just says which observation each one was within its group. The interesting things are `treatment` and `sleeptime`, which are the two variables we'll need for our analysis of variance.

Sometimes the column playing the role of *rep* *is* interesting to us, but not here.

(c) Using your new data frame, make side-by-side boxplots of sleep time by treatment group.

Solution

```
ggplot(sleep, aes(x = treatment, y = sleeptime)) +
    geom_boxplot()
```

(d) In your boxplots, how does the median sleep time appear to depend on treatment group?

Solution

It appears to *decrease* as the dose of ethanol increases, and pretty substantially so (in that the differences ought to be significant, but that's coming up).

(e) There is an assumption about spread that the analysis of variance needs in order to be reliable. Do your boxplots indicate that this assumption is satisfied for these data, bearing in mind that you have only five observations per group?

Solution

The assumption is that the population SDs of each group are all equal. Now, the boxplots show IQRs, which are kind of a surrogate for SD, and because we only have five observations per group to base the IQRs on, the *sample* IQRs might vary a bit. So we should look at the heights of the boxes on the boxplot, and see whether they are grossly unequal. They appear to be to be of very similar heights, all things considered, so I am happy. If you want the SDs themselves:

```
sleep %>% group_by(treatment) %>% summarize(stddev = sd(sleeptime))
```

```
## # A tibble: 4 x 2
##   treatment stddev
##   <chr>      <dbl>
## 1 e0         10.2
## 2 e1          9.34
## 3 e2          9.46
## 4 e4          9.56
```

Those are *very* similar, given only 5 observations per group. No problems here.

(f) Run an analysis of variance to see whether sleep time differs significantly among treatment groups. What do you conclude?

Solution

I use `aov` here, because I might be following up with Tukey in a minute:

```
sleep.1 = aov(sleeptime ~ treatment, data = sleep)
summary(sleep.1)
```

```
##             Df Sum Sq Mean Sq F value
## treatment    3   5882    1961   21.09
## Residuals   16   1487      93
##               Pr(>F)
## treatment   8.32e-06 ***
## Residuals
## ---
## Signif. codes:
##   0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1  ' ' 1
```

This is a very small P-value, so my conclusion is that the mean sleep times are not all the same for the treatment groups. Further than that I am not entitled to say (yet).

The technique here is to save the output from `aov` in something, look at that (via `summary`), and then that same something gets fed into `TukeyHSD` later.

(g) Would it be a good idea to run Tukey's method here? Explain briefly why or why not, and if you think it would be a good idea, run it.

Solution

Tukey's method is useful when (i) we have run an analysis of variance and got a significant result and (ii) when we want to know which groups differ significantly from which. Both (i) and (ii) are true here. So:

```
TukeyHSD(sleep.1)
```

```
##   Tukey multiple comparisons of means
##     95% family-wise confidence level
##
## Fit: aov(formula = sleeptime ~ treatment, data = sleep)
##
## $treatment
##         diff       lwr         upr     p adj
## e1-e0 -17.74 -35.18636  -0.2936428 0.0455781
## e2-e0 -31.36 -48.80636 -13.9136428 0.0005142
## e4-e0 -46.52 -63.96636 -29.0736428 0.0000056
## e2-e1 -13.62 -31.06636   3.8263572 0.1563545
## e4-e1 -28.78 -46.22636 -11.3336428 0.0011925
## e4-e2 -15.16 -32.60636   2.2863572 0.1005398
```

(h) What do you conclude from Tukey's method? (This is liable to be a bit complicated.) Is there a treatment that is clearly best, in terms of the sleep time being largest?

Solution

All the differences are significant except treatment `e2` vs. `e1` and `e4`. All the differences involving the control group `e0` are significant, and if you look back at the boxplots in (c), you'll see that the control group `e0` had the *highest* mean sleep time. So the control group is best (from this point of view), or another way of saying it is that *any* dose of ethanol is significantly reducing mean sleep time. The other comparisons are a bit confusing, because the 1-4 difference is significant, but neither of the differences involving 2 are. That is, 1 is

better than 4, but 2 is not significantly worse than 1 nor better than 4. This seems like it should be a logical impossibility, but the story is that we don't have enough data to decide where 2 fits relative to 1 or 4. If we had 10 or 20 observations per group, we might be able to conclude that 2 is in between 1 and 4 as the boxplots suggest.

12.4 *Growth of tomatoes*

A biology graduate student exposed each of 32 tomato plants to one of four different colours of light (8 plants to each colour). The growth rate of each plant, in millimetres per week, was recorded. The data are in link.

(a) Read the data into R and confirm that you have 8 rows and 5 columns of data.

Solution
This kind of thing:

```
my_url = "http://www.utsc.utoronto.ca/~butler/c32/tomatoes.txt"
toms1 = read_delim(my_url, " ")
```

```
## Parsed with column specification:
## cols(
##   plant = col_integer(),
##   blue = col_double(),
##   red = col_double(),
##   yellow = col_double(),
##   green = col_double()
## )
```

```
toms1
```

```
## # A tibble: 8 x 5
##   plant  blue   red yellow green
##   <int> <dbl> <dbl>  <dbl> <dbl>
## 1     1  5.34  13.7   4.61  2.72
## 2     2  7.45  13.0   6.63  1.08
## 3     3  7.15  10.2   5.29  3.97
## 4     4  5.53  13.1   5.29  2.66
## 5     5  6.34  11.1   4.76  3.69
## 6     6  7.16  11.4   5.57  1.96
## 7     7  7.77  14.0   6.57  3.38
## 8     8  5.09  13.5   5.25  1.87
```

I do indeed have 8 rows and 5 columns.
With only 8 rows, listing the data like this is good.

(b) Re-arrange the data so that you have *one* column containing all the growth rates, and another column saying which colour light each plant was exposed to. (The aim here is to produce something suitable for feeding into `aov` later.)

Solution
This is a job for `gather`:

```
toms2 = toms1 %>% gather(colour, growthrate, blue:green)
toms2
```

```
## # A tibble: 32 x 3
##    plant colour growthrate
##    <int> <chr>       <dbl>
##  1     1 blue         5.34
##  2     2 blue         7.45
##  3     3 blue         7.15
##  4     4 blue         5.53
##  5     5 blue         6.34
##  6     6 blue         7.16
##  7     7 blue         7.77
##  8     8 blue         5.09
##  9     1 red         13.7
## 10     2 red         13.0
## # ... with 22 more rows
```

Reminder: data frame to gather, what makes the columns different (they're different colours), what makes them the same (they're all growth rates), which columns to gather together (all the colour ones).

Since the column `plant` was never mentioned, this gets repeated as necessary, so now it denotes "plant within colour group", which in this case is not very useful. (Where you have matched pairs, or repeated measures in general, you *do* want to keep track of which individual is which. But this is not repeated measures because plant number 1 in the blue group and plant number 1 in the red group are *different* plants.)

There were 8 rows originally and 4 different colours, so there should be, and are, $8 \times 4 = 32$ rows in the gathered-up data set.

(c) Save the data in the new format to a text file. This is most easily done using `write_csv`, which is the opposite of `read_csv`. It requires two things: a data frame, and the name of a file to save in, which should have a `.csv` extension.

Solution
The code is easy enough:

```
write_csv(toms2, "tomatoes2.csv")
```

If no error, it worked. That's all you need.

To verify (for my satisfaction) that it was saved correctly:

```
cat tomatoes2.csv
```

```
## plant,colour,growthrate
## 1,blue,5.34
## 2,blue,7.45
## 3,blue,7.15
## 4,blue,5.53
## 5,blue,6.34
## 6,blue,7.16
## 7,blue,7.77
## 8,blue,5.09
## 1,red,13.67
## 2,red,13.04
## 3,red,10.16
## 4,red,13.12
## 5,red,11.06
## 6,red,11.43
## 7,red,13.98
## 8,red,13.49
## 1,yellow,4.61
## 2,yellow,6.63
## 3,yellow,5.29
## 4,yellow,5.29
## 5,yellow,4.76
## 6,yellow,5.57
## 7,yellow,6.57
## 8,yellow,5.25
## 1,green,2.72
## 2,green,1.08
## 3,green,3.97
## 4,green,2.66
## 5,green,3.69
## 6,green,1.96
## 7,green,3.38
## 8,green,1.87
```

On my system, that will list the contents of the file. Or you can just open it in R Studio (if you saved it the way I did, it'll be in the same folder, and you can find it in the Files pane.)

(d) Make a suitable boxplot, and use it to assess the assumptions for ANOVA. What do you conclude? Explain briefly.

Solution

Nothing terribly surprising here. My data frame is called `toms2`, for some reason:

```
ggplot(toms2, aes(x = colour, y = growthrate)) +
    geom_boxplot()
```

There are no outliers, but there is a little skewness (compare the *whiskers*, not the placement of the median within the box, because what matters with skewness is the *tails*, not the middle of the distribution; it's problems in the tails that make the mean unsuitable as a measure of centre). The Red group looks the most skewed. Also, the Yellow group has smaller spread than the others (we assume that the population variances within each group are equal). The thing to bear in mind here, though, is that there are only eight observations per group, so the distributions could appear to have unequal variances or some non-normality by chance.

My take is that these data, all things considered, are *just about* OK for ANOVA. Another option would be to do Welch's ANOVA as well and compare with the regular ANOVA: if they give more or less the same P-value, that's a sign that I didn't need to worry.

Extra: some people like to run a formal test on the variances to test them for equality. My favourite (for reasons explained elsewhere) is the Levene test, if you insist on going this way. It lives in package `car`, and *does not* take a `data=`, so you need to do the `with` thing:

```
library(car)

## Loading required package: carData

##
## Attaching package: 'car'
```

```
## The following object is masked from 'package:dplyr':
##
##     recode

## The following object is masked from 'package:purrr':
##
##     some
```

```
with(toms2, leveneTest(growthrate, colour))
```

```
## Warning in leveneTest.default(growthrate,
## colour): colour coerced to factor.

## Levene's Test for Homogeneity of Variance (center = median)
##       Df F value Pr(>F)
## group  3  0.9075 0.4499
##       28
```

The warning is because `colour` was actually text, but the test did the right thing by turning it into a factor, so that's OK.

There is no way we can reject equal variances in the four groups. The F-statistic is less than 1, in fact, which says that if the four groups have the same population variances, the sample variances will be *more* different than the ones we observed on average, and so there is no way that these sample variances indicate different population variances. (This is because of 8 observations only per group; if there had been 80 observations per group, it would have been a different story.) Decide for yourself whether you're surprised by this.

With that in mind, I think the regular ANOVA will be perfectly good, and we would expect that and the Welch ANOVA to give very similar results.

I don't need `car` again, so let's get rid of it:

```
detach("package:car", unload = T)
```

(e) Run (regular) ANOVA on these data. What do you conclude? (Optional extra: if you think that some other variant of ANOVA would be better, run that as well and compare the results.)

Solution

`aov`, bearing in mind that Tukey is likely to follow:

```
toms.1 = aov(growthrate ~ colour, data = toms2)
summary(toms.1)
```

```
##             Df Sum Sq Mean Sq F value
## colour       3  410.5  136.82   118.2
```

```
## Residuals   28   32.4    1.16
##               Pr(>F)
## colour      5.28e-16 ***
## Residuals
## ---
## Signif. codes:
##   0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1  ' ' 1
```

This is a tiny P-value, so the mean growth rate for the different colours is definitely *not* the same for all colours. Or, if you like, one or more of the colours has a different mean growth rate than the others.

This, remember, is as far as we go right now.

Extra: if you thought that normality was OK but not equal spreads, then Welch ANOVA is the way to go:

```
toms.2 = oneway.test(growthrate ~ colour, data = toms2)
toms.2
```

```
##
##  One-way analysis of means (not
##  assuming equal variances)
##
## data:  growthrate and colour
## F = 81.079, num df = 3.000, denom df =
## 15.227, p-value = 1.377e-09
```

The P-value is not *quite* as small as for the regular ANOVA, but it is still very small, and the conclusion is the same.

If you had doubts about the normality (that were sufficiently great, even given the small sample sizes), then go with Mood's median test for multiple groups:

```
library(smmr)
median_test(toms2, growthrate, colour)
```

```
## $table
##         above
## group    above below
##   blue       5     3
##   green      0     8
##   red        8     0
##   yellow     3     5
##
## $test
##        what        value
```

```
## 1 statistic 1.700000e+01
## 2        df 3.000000e+00
## 3   P-value 7.067424e-04
```

The P-value is again extremely small (though not quite as small as for the other two tests, for the usual reason that Mood's median test doesn't use the data very efficiently: it doesn't use how *far* above or below the overall median the data values are.)

The story here, as ever, is consistency: whatever you thought was wrong, looking at the boxplots, needs to guide the test you do. This should probably be a flow chart, but this way works too:

- if you are not happy with normality, go with `median_test` from `smmr` (Mood's median test).
- if you are happy with normality and equal variances, go with `aov`.
- if you are happy with normality but not equal variances, go with `oneway.test` (Welch ANOVA).

So the first thing to think about is normality, and if you are OK with normality, then think about equal spreads. Bear in mind that you need to be willing to tolerate a certain amount of non-normality and inequality in the spreads, given that your data are only samples from their populations. (Don't expect perfection, in short.)

(f) If warranted, run a suitable follow-up. (If not warranted, explain briefly why not.)

Solution

Whichever flavour of ANOVA you ran (regular ANOVA, Welch ANOVA, Mood's median test), you got the same conclusion for these data: that the average growth rates were not all the same for the four colours. That, as you'll remember, is as far as you go. To find out which colours differ from which in terms of growth rate, you need to run some kind of multiple-comparisons follow-up, the right one for the analysis you did. Looking at the boxplots suggests that red is clearly best and green clearly worst, and it is possible that all the colours are significantly different from each other.) If you did regular ANOVA, Tukey is what you need:

```
TukeyHSD(toms.1)
```

```
##   Tukey multiple comparisons of means
##     95% family-wise confidence level
##
## Fit: aov(formula = growthrate ~ colour, data = toms2)
```

```
##
## $colour
##                 diff       lwr        upr
## green-blue   -3.8125 -5.281129 -2.3438706
## red-blue      6.0150  4.546371  7.4836294
## yellow-blue  -0.9825 -2.451129  0.4861294
## red-green     9.8275  8.358871 11.2961294
## yellow-green  2.8300  1.361371  4.2986294
## yellow-red   -6.9975 -8.466129 -5.5288706
##                  p adj
## green-blue   0.0000006
## red-blue     0.0000000
## yellow-blue  0.2825002
## red-green    0.0000000
## yellow-green 0.0000766
## yellow-red   0.0000000
```

All of the differences are (strongly) significant, except for yellow and blue, the two with middling growth rates on the boxplot. Thus we would have no hesitation in saying that growth rate is biggest in red light and smallest in green light.

If you did Welch ANOVA, you need Games-Howell, which you have to get from one of the packages that offers it:

```
library(PMCMRplus)
gamesHowellTest(growthrate ~ factor(colour), data = toms2)
```

```
##
##  Pairwise comparisons using Games-Howell test

## data: growthrate by factor(colour)

##        blue    green   red
## green  1.6e-05 -       -
## red    1.5e-06 4.8e-09 -
## yellow 0.18707 0.00011 5.8e-07

##
## P value adjustment method: none

## alternative hypothesis: two.sided
```

The conclusions are the same as for the Tukey: all the means are significantly different except for yellow and blue. Finally, if you did Mood's median test, you need this one:

```
pairwise_median_test(toms2, growthrate, colour)
```

```
## # A tibble: 6 x 4
##   g1    g2       p_value adj_p_value
##   <chr> <chr>      <dbl>       <dbl>
## 1 blue  green  0.0000633    0.000380
## 2 blue  red    0.0000633    0.000380
## 3 blue  yellow 0.317        1.90
## 4 green red    0.0000633    0.000380
## 5 green yellow 0.0000633    0.000380
## 6 red   yellow 0.0000633    0.000380
```

Same conclusions again. This is what I would have guessed; the conclusions from Tukey were so clear-cut that it really didn't matter which way you went; you'd come to the same conclusion.

That said, what I am looking for from you is a sensible choice of analysis of variance (ANOVA, Welch's ANOVA or Mood's median test) for a good reason, followed by the *right* follow-up for the test you did. Even though the conclusions are all the same no matter what you do here, I want you to get used to following the right method, so that you will be able to do the right thing when it *does* matter.

12.5 *Pain relief in migraine headaches (again)*

The data in link are from a study of pain relief in migraine headaches. Specifically, 27 subjects were randomly assigned to receive *one* of three pain relieving drugs, labelled A, B and C. Each subject reported the number of hours of pain relief they obtained (that is, the number of hours between taking the drug and the migraine symptoms returning). A higher value is therefore better. Can we make some recommendation about which drug is best for the population of migraine sufferers?

(a) Read in and display the data. Take a look at the data file first, and see if you can say why `read_table` will work and `read_delim` will not.

Solution

The key is two things: the data values are *lined up in columns*, and *there is more than one space between values*. The second thing is why `read_delim` will not work. If you look carefully at the data file, you'll see that the column names are above and aligned with the columns, which is what `read_table` wants. If the column names had *not* been aligned with the columns, we would have needed `read_table2`.

```
my_url = "http://www.utsc.utoronto.ca/~butler/c32/migraine.txt"
migraine = read_table(my_url)
```

```
## Parsed with column specification:
## cols(
##   DrugA = col_integer(),
##   DrugB = col_integer(),
##   DrugC = col_integer()
## )
```

```
migraine
```

```
## # A tibble: 9 x 3
##   DrugA DrugB DrugC
##   <int> <int> <int>
## 1     4     6     6
## 2     5     8     7
## 3     4     4     6
## 4     3     5     6
## 5     2     4     7
## 6     4     6     5
## 7     3     5     6
## 8     4    11     5
## 9     4    10     5
```

Success.

(b) What is it about the experimental design that makes a one-way analysis of variance plausible for data like this?

Solution

Each experimental subject only tested *one* drug, so that we have 27 independent observations, nine from each drug. This is exactly the setup that a one-way ANOVA requires. Compare that to, for example, a situation where you had only 9 subjects, but they each tested *all* the drugs (so that each subject produced three measurements). That is like a three-measurement version of matched pairs, a so-called **repeated-measures design**, which requires its own kind of analysis.

To allow for the fact that measurements on the same subject are not independent but correlated.

(c) What is wrong with the current format of the data as far as doing a one-way ANOVA analysis is concerned? (This is related to the idea of whether or not the data are "tidy".)

Solution

For our analysis, we need one column of pain relief time and one column labelling the drug that the subject in question took. Or, if you prefer to think about what would make these data "tidy": there are 27 subjects, so there ought to be 27 rows, and all three columns are measurements of pain relief, so they ought to be in one column.

(d) "Tidy" the data to produce a data frame suitable for your analysis.

Solution

`gather` the columns that are all measurements of one thing. The syntax of `gather` is: what makes the columns different, what makes them the same, and which columns need to be gathered together. Use a pipe to name the dataframe to work with. I'm going to save my new data frame:

```
(migraine2 <- migraine %>% gather(drug, painrelief,
    DrugA:DrugC))
```

```
## # A tibble: 27 x 2
##    drug  painrelief
##    <chr>      <int>
##  1 DrugA          4
##  2 DrugA          5
##  3 DrugA          4
##  4 DrugA          3
##  5 DrugA          2
##  6 DrugA          4
##  7 DrugA          3
##  8 DrugA          4
##  9 DrugA          4
## 10 DrugB          6
## # ... with 17 more rows
```

The brackets around the whole thing print out the result as well as saving it. If you don't have those, you'll need to type `migraine2` again to display it.

We do indeed have a new data frame with 27 rows, one per observation, and 2 columns, one for each variable: the pain relief hours, plus a column identifying which drug that pain relief time came from. Exactly what `aov` needs.

You can probably devise a better name for your new data frame.

(e) Go ahead and run your one-way ANOVA (and Tukey if necessary). Assume for this that the pain relief hours in each group are sufficiently close to normally distributed with sufficiently equal spreads.

Solution

My last sentence absolves us from doing the boxplots that we would normally insist on doing.

```
painrelief.1 = aov(painrelief ~ drug, data = migraine2)
summary(painrelief.1)
```

```
##             Df Sum Sq Mean Sq F value
## drug         2  41.19   20.59   7.831
## Residuals   24  63.11    2.63
##              Pr(>F)
## drug        0.00241 **
## Residuals
## ---
## Signif. codes:
##   0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1  ' ' 1
```

There are (strongly) significant differences among the drugs, so it is definitely worth firing up Tukey to figure out where the differences are:

```
TukeyHSD(painrelief.1)
```

```
##   Tukey multiple comparisons of means
##     95% family-wise confidence level
##
## Fit: aov(formula = painrelief ~ drug, data = migraine2)
##
## $drug
##                   diff        lwr      upr
## DrugB-DrugA  2.8888889  0.9798731 4.797905
## DrugC-DrugA  2.2222222  0.3132065 4.131238
## DrugC-DrugB -0.6666667 -2.5756824 1.242349
##                 p adj
## DrugB-DrugA 0.0025509
## DrugC-DrugA 0.0203671
## DrugC-DrugB 0.6626647
```

Both the differences involving drug A are significant, and because a high value of `painrelief` is better, in both cases drug A is *worse* than the other drugs. Drugs B and C are not significantly different from each other.

Extra: we can also use the "pipe" to do this all in one go:

```
migraine %>% gather(drug, painrelief, DrugA:DrugC) %>%
    aov(painrelief ~ drug, data = .) %>% summary()
```

```
##             Df Sum Sq Mean Sq F value
## drug         2  41.19   20.59   7.831
## Residuals   24  63.11    2.63
##              Pr(>F)
## drug        0.00241 **
## Residuals
```

```
## ---
## Signif. codes:
##   0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1  ' ' 1
```

with the same results as before. Notice that I never actually created a second data frame by name; it was created by `gather` and then immediately used as input to `aov`. I also used the `data=.` trick to use "the data frame that came out of the previous step" as my input to `aov`.

And then thrown away.

Read the above like this: "take `migraine`, and then gather together the `DrugA` through `DrugC` columns into a column `painrelief`, labelling each by its drug, and then do an ANOVA of `painrelief` by `drug`, and then summarize the results."

What is even more alarming is that I can feed the output from `aov` straight into `TukeyHSD`:

```
migraine %>% gather(drug, painrelief, DrugA:DrugC) %>%
    aov(painrelief ~ drug, data = .) %>% TukeyHSD()
```

```
##   Tukey multiple comparisons of means
##     95% family-wise confidence level
##
## Fit: aov(formula = painrelief ~ drug, data = .)
##
## $drug
##                   diff        lwr      upr
## DrugB-DrugA  2.8888889  0.9798731 4.797905
## DrugC-DrugA  2.2222222  0.3132065 4.131238
## DrugC-DrugB -0.6666667 -2.5756824 1.242349
##                 p adj
## DrugB-DrugA 0.0025509
## DrugC-DrugA 0.0203671
## DrugC-DrugB 0.6626647
```

I wasn't sure whether this would work, since the output from `aov` is an R `list` rather than a data frame, but the output from `aov` is sent into `TukeyHSD` whatever kind of thing it is.

What I am missing here is to display the result of `aov` *and* use it as input to `TukeyHSD`. Of course, I had to discover that this could be solved, and indeed it can:

```
migraine %>% gather(drug, painrelief, DrugA:DrugC) %>%
    aov(painrelief ~ drug, data = .) %>% {
    print(summary(.))
    .
} %>% TukeyHSD()
```

```
##             Df Sum Sq Mean Sq F value
## drug         2  41.19   20.59   7.831
## Residuals   24  63.11    2.63
##              Pr(>F)
## drug        0.00241 **
## Residuals
## ---
## Signif. codes:
##   0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1  ' ' 1

##   Tukey multiple comparisons of means
##     95% family-wise confidence level
##
## Fit: aov(formula = painrelief ~ drug, data = .)
##
## $drug
##                   diff        lwr      upr
## DrugB-DrugA  2.8888889  0.9798731 4.797905
## DrugC-DrugA  2.2222222  0.3132065 4.131238
## DrugC-DrugB -0.6666667 -2.5756824 1.242349
##                 p adj
## DrugB-DrugA 0.0025509
## DrugC-DrugA 0.0203671
## DrugC-DrugB 0.6626647
```

The odd-looking second-last line of that uses that `.` trick for "whatever came out of the previous step". The thing inside the curly brackets is two commands one after the other; the first is to display the `summary` of that `aov` and the second part after the `;` is to just pass whatever came out of the previous line, the output from `aov`, on, unchanged, into `TukeyHSD`.

It needs *print* around it to display it, as you need *print* to display something within a loop or a function.

In the Unix world this is called `tee`, where you print something *and* pass it on to the next step. The name `tee` comes from a (real) pipe that plumbers would use to split water flow into two, which looks like a letter T.

(f) What recommendation would you make about the best drug or drugs? Explain briefly.

Solution

Drug A is significantly the worst, so we eliminate that. But there is no significant difference between drugs B and C, so we have no reproducible reason for preferring one rather than the other. Thus, we recommend "either B or C". If you weren't sure which way around the drugs actually came out, then you should work out the mean pain relief score by drug:

```
migraine2 %>% group_by(drug) %>% summarize(m = mean(painrelief))
```

```
## # A tibble: 3 x 2
##   drug      m
##   <chr> <dbl>
## 1 DrugA  3.67
## 2 DrugB  6.56
## 3 DrugC  5.89
```

These confirm that A is worst, and there is nothing much to choose between B and C. You should *not* recommend drug C over drug B on this evidence, just because its (sample) mean is higher than B's. The point about significant differences is that they are supposed to stand up to replication: in another experiment, or in real-life experiences with these drugs, the mean pain relief score for drug A is expected to be worst, but between drugs B and C, sometimes the mean of B will come out higher and sometimes C's mean will be higher, because there is no significant difference between them. Another way is to draw a boxplot of pain-relief scores:

This talks about *means* rather than individual observations; in individual cases, sometimes even drug *A* will come out best. But we're interested in population means, since we want to do the greatest good for the greatest number. *Greatest good for the greatest number* is from Jeremy Bentham, 1748–1832, British philosopher and advocate of utilitarianism.

```
ggplot(migraine2, aes(x = drug, y = painrelief)) +
    geom_boxplot()
```

The medians of drugs B and C are actually exactly the same. Because the pain relief values are all whole numbers (and there are only 9 in each group), you get that thing where enough of them are equal that the median and third quartiles are equal, actually for all three groups.

Despite the outlier, I'm willing to call these groups sufficiently symmetric for the ANOVA to be OK (but I didn't ask you to draw the boxplot, because I didn't want to confuse the issue with this. The point of this question was to get the data tidy enough to do an

analysis.) Think about it for a moment: that outlier is a value of 8. This is really not that much bigger than the value of 7 that is the highest one on drug C. The 7 for drug C is not an outlier. The only reason the 8 came out as an outlier was because the IQR was only 1. If the IQR on drug B had happened to be a bit bigger, the 8 would not have been an outlier.

As I said, I didn't want you to have to get into this, but if you are worried, you know what the remedy is — Mood's median test. Don't forget to use the right data frame:

```
library(smmr)
median_test(migraine2, painrelief, drug)
```

```
## $table
##        above
## group   above below
##   DrugA     0     8
##   DrugB     5     2
##   DrugC     6     0
##
## $test
##        what        value
## 1 statistic 1.527273e+01
## 2        df 2.000000e+00
## 3   P-value 4.825801e-04
```

Because the pain relief scores are integers, there are probably a lot of them equal to the overall median. There were 27 observations altogether, but Mood's median test will discard any that are equal to this value. There must have been 9 observations in each group to start with, but if you look at each row of the table, there are only 8 observations listed for drug A, 7 for drug B and 6 for drug C, so there must have been 1, 2 and 3 (totalling 6) observations equal to the median that were discarded.

The P-value is a little bigger than came out of the F-test, but the conclusion is still that there are definitely differences among the drugs in terms of pain relief. The table at the top of the output again suggests that drug A is worse than the others, but to confirm that you'd have to do Mood's median test on all three *pairs* of drugs, and then use Bonferroni to allow for your having done three tests:

```
pairwise_median_test(migraine2, painrelief, drug)
```

```
## # A tibble: 3 x 4
##   g1    g2     p_value adj_p_value
##   <chr> <chr>    <dbl>       <dbl>
```

```
## 1 DrugA DrugB 0.00721     0.0216
## 2 DrugA DrugC 0.000183    0.000548
## 3 DrugB DrugC 0.921       2.76
```

Drug A gives worse pain relief (fewer hours) than both drugs B and C, which are not significantly different from each hour. This is exactly what you would have guessed from the boxplot.

I gotta do something about those adjusted P-values bigger than 1!

12.6 *Location, species and disease in plants*

The table below is a "contingency table", showing frequencies of diseased and undiseased plants of two different species in two different locations:

```
Species     Disease present         Disease absent
Location X Location Y  Location X Location Y
A            44         12          38        10
B            28         22          20        18
```

The data were saved as link. In that file, the columns are coded by two letters: a `p` or an `a` to denote presence or absence of disease, and an `x` or a `y` to denote location X or Y. The data are separated by multiple spaces and aligned with the variable names.

(a) Read in and display the data.

Solution

`read_table` again. You know this because, when you looked at the data file, which of course you did (didn't you?), you saw that the data values were aligned by columns with multiple spaces between them:

```
my_url = "http://www.utsc.utoronto.ca/~butler/c32/disease.txt"
tbl = read_table(my_url)
```

```
## Parsed with column specification:
## cols(
##   Species = col_character(),
##   px = col_integer(),
##   py = col_integer(),
##   ax = col_integer(),
##   ay = col_integer()
## )
```

```
tbl
```

```
## # A tibble: 2 x 5
##   Species    px    py    ax    ay
##   <chr>   <int> <int> <int> <int>
## 1 A          44    12    38    10
## 2 B          28    22    20    18
```

I was thinking ahead, since I'll be wanting to have one of my columns called `disease`, so I'm *not* calling the data frame `disease`.

You'll also have noticed that I simplified the data frame that I had you read in, because the original contingency table I showed you has *two* header rows, and we have to have *one* header row. So I mixed up the information in the two header rows into one.

(b) Explain briefly how these data are not "tidy".

Solution

The simple answer is that there are 8 frequencies, that each ought to be in a row by themselves. Or, if you like, there are three variables, Species, Disease status and Location, and each of *those* should be in a *column* of its own. Either one of these ideas, or something like it, is good. I need you to demonstrate that you know something about "tidy data" in this context.

(c) Use a suitable `tidyr` tool to get all the things that are the same into a single column. (You'll need to make up a temporary name for the other new column that you create.) Show your result.

Solution

`gather` is the tool. All the columns apart from `Species` contain frequencies, so that's what's "the same". They are frequencies in disease-location combinations, so I'll call the column of "what's different" `disloc`. Feel free to call it `temp` for now if you prefer:

```
(tbl.2 <- tbl %>% gather(disloc, frequency, px:ay))
```

```
## # A tibble: 8 x 3
##   Species disloc frequency
##   <chr>   <chr>      <int>
## 1 A       px            44
## 2 B       px            28
## 3 A       py            12
## 4 B       py            22
## 5 A       ax            38
## 6 B       ax            20
## 7 A       ay            10
## 8 B       ay            18
```

This also works ("gather together everything but `Species`"):

```
(tbl.2 <- tbl %>% gather(disloc, frequency, -Species))
```

```
## # A tibble: 8 x 3
##   Species disloc frequency
##   <chr>   <chr>      <int>
## 1 A       px            44
## 2 B       px            28
## 3 A       py            12
## 4 B       py            22
## 5 A       ax            38
## 6 B       ax            20
## 7 A       ay            10
## 8 B       ay            18
```

(d) Explain briefly how the data frame you just created is still not "tidy" yet.

Solution

The column I called `disloc` actually contains *two* variables, disease and location, which need to be split up. A check on this is that we have two columns (not including the frequencies), but back in (b) we found *three* variables, so there ought to be three non-frequency columns.

(e) Use one more `tidyr` tool to make these data tidy, and show your result.

Solution

This means splitting up `disloc` into two separate columns, splitting after the first character, thus:

```
(tbl.3 <- tbl.2 %>% separate(disloc, c("disease",
    "location"), 1))
```

```
## # A tibble: 8 x 4
##   Species disease location frequency
##   <chr>   <chr>   <chr>        <int>
## 1 A       p       x               44
## 2 B       p       x               28
## 3 A       p       y               12
## 4 B       p       y               22
## 5 A       a       x               38
## 6 B       a       x               20
## 7 A       a       y               10
## 8 B       a       y               18
```

This is now tidy: eight frequencies in rows, and three non-frequency columns. (Go back and look at your answer to part (b) and note that the issues you found there have all been resolved now.)

(f) Let's see if we can re-construct the original contingency table (or something equivalent to it). Use the function `xtabs`. This requires first a model formula with the frequency variable on the left of the squiggle, and the other variables separated by plus signs on the right. Second it requires a data frame, with `data=`. Feed your data frame from the previous part into `xtabs`. Save the result in a variable and display the result.

Solution

```
tbl.4 = xtabs(frequency ~ Species + disease +
    location, data = tbl.3)
tbl.4
```

```
## , , location = x
##
##        disease
## Species  a  p
##       A 38 44
##       B 20 28
##
## , , location = y
##
##        disease
## Species  a  p
##       A 10 12
##       B 18 22
```

This shows a pair of contingency tables, one each for each of the two locations (in general, the variable you put last on the right side of the model formula). You can check that everything corresponds with the original data layout at the beginning of the question, possibly with some things rearranged (but with the same frequencies in the same places).

(g) Take the output from the last part and feed it into the function `ftable`. How has the output been changed? Which do you like better? Explain briefly.

Solution
This:

```
ftable(tbl.4)
```

```
##                 location  x  y
## Species disease
## A       a                38 10
##         p                44 12
## B       a                20 18
##         p                28 22
```

This is the same output, but shown more compactly. (Rather like a vertical version of the original data, in fact.) I like `ftable` better because it displays the data in the smallest amount of space, though I'm fine if you prefer the `xtabs` output because it spreads things out more. This is a matter of taste. Pick one and tell me why you prefer it, and I'm good.

That's the end of what you had to do, but I thought I would do some modelling and try to find out what's associated with disease. The appropriate modelling with frequencies is called "log-linear modelling", and it assumes that the log of the frequencies has a linear relationship with the effects of the other variables. This is not quite as simple as the log transformations we had before, because bigger frequencies are going to be more variable, so we fit a generalized linear model with a Poisson-distributed response and log link. (It's better if you know what that means, but you ought to be able to follow the logic if you don't.)

First, fit a model predicting frequency from everything, including all the interactions. (The reason for doing it this way will become clear later):

```r
model.1 = glm(frequency ~ Species * location *
    disease, data = tbl.3, family = "poisson")
drop1(model.1, test = "Chisq")
```

```
## Single term deletions
##
## Model:
## frequency ~ Species * location * disease
##                          Df Deviance    AIC
## <none>                      0.000000 55.291
## Species:location:disease  1 0.070257 53.362
##                              LRT Pr(>Chi)
## <none>
## Species:location:disease 0.070257    0.791
```

The residuals are all zero because this model fits perfectly. The problem is that it is very complicated, so it offers no insight. So what we do is to look at the highest-order interaction `Species:location:disease` and see whether it is significant. It is not, so we can remove it. This

is reminiscent of variable selection in regression, where we pull the least significant thing out of the model in turn until we can go no further. But here, we have additional things to think about: we have to get rid of all the three-way interactions before we can tackle the two-way ones, and all the two-way ones before we can tackle the main effects. There is a so-called "nested" structure happening here that says you don't look at, say, `Species`, until you have removed *all* the higher-order interactions involving `Species`. Not clear yet? Don't fret. `drop1` allows you to assess what is currently up for grabs (here, only the three-way interaction, which is not significant, so out it comes).

Let's get rid of that three-way interaction. This is another use for `update` that we've seen in connection with multiple regression (to make small changes to a big model):

```
model.2 = update(model.1, . ~ . - Species:location:disease)
drop1(model.2, test = "Chisq")
```

```
## Single term deletions
##
## Model:
## frequency ~ Species + location + disease + Species:location +
##     Species:disease + location:disease
##                  Df Deviance    AIC     LRT
## <none>                0.0703 53.362
## Species:location  1  13.0627 64.354 12.9924
## Species:disease   1   0.2696 51.561  0.1993
## location:disease  1   0.1043 51.396  0.0340
##                   Pr(>Chi)
## <none>
## Species:location 0.0003128 ***
## Species:disease  0.6552865
## location:disease 0.8536877
## ---
## Signif. codes:
##   0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1  ' ' 1
```

Notice how `update` saved us having to write the whole model out again.

Now the three two-way interactions are up for grabs: `Species:location`, `Species:disease` and `location:disease`. The last of these is the least significant, so out it comes. I did some copying and pasting, but I had to remember which model I was working with and what I was removing:

```
model.3 = update(model.2, . ~ . - location:disease)
drop1(model.3, test = "Chisq")
```

```
## Single term deletions
##
## Model:
## frequency ~ Species + location + disease + Species:location +
##     Species:disease
##                  Df Deviance    AIC     LRT
## <none>                0.1043 51.396
## Species:location  1  13.0678 62.359 12.9635
## Species:disease   1   0.2746 49.566  0.1703
##                   Pr(>Chi)
## <none>
## Species:location 0.0003176 ***
## Species:disease  0.6798021
## ---
## Signif. codes:
##   0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1  ' ' 1
```

`Species:disease` comes out, but it looks as if `Species:location` will have to stay:

```r
model.4 = update(model.3, . ~ . - Species:disease)
drop1(model.4, test = "Chisq")
```

```
## Single term deletions
##
## Model:
## frequency ~ Species + location + disease + Species:location
##                  Df Deviance    AIC     LRT
## <none>                0.2746 49.566
## disease           1   2.3617 49.653  2.0871
## Species:location  1  13.2381 60.530 12.9635
##                   Pr(>Chi)
## <none>
## disease          0.1485461
## Species:location 0.0003176 ***
## ---
## Signif. codes:
##   0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1  ' ' 1
```

`Species:location` indeed stays. That means that anything "contained in" it also has to stay, regardless of its main effect. So the only candidate for removal now is `disease`: not significant, out it comes:

```r
model.5 = update(model.4, . ~ . - disease)
drop1(model.5, test = "Chisq")
```

```
## Single term deletions
##
## Model:
## frequency ~ Species + location + Species:location
##                  Df Deviance    AIC    LRT
## <none>                2.3617 49.653
## Species:location  1  15.3252 60.617 12.963
##                   Pr(>Chi)
## <none>
## Species:location 0.0003176 ***
## ---
## Signif. codes:
##   0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1  ' ' 1
```

And now we have to stop.

What does this final model mean? Well, frequency depends significantly on the `Species:location` combination, but not on anything else. To see how, we make a contingency table of species by location (totalling up over disease status, since that is not significant):

```r
xtabs(frequency ~ Species + location, data = tbl.3)
```

```
##        location
## Species  x  y
##       A 82 22
##       B 48 40
```

Most of the species A's are at location X, but the species B's are about evenly divided between the two locations. Or, if you prefer (equally good): location X has mostly species A, while location Y has mostly species B. You can condition on either variable and compare the conditional distribution of the other one.

Now, this is rather interesting, because this began as a study of disease, but disease has completely disappeared from our final model! That means that nothing in our final model has any relationship with disease. Indeed, if you check the original table, you'll find that disease is present slightly more than it's absent, for all combinations of species and location. That is, neither species nor location has any particular association with (effect on) disease, since disease prevalence doesn't change appreciably if you change location, species or the combination of them.

The way an association with disease would show up is if a `disease:`something interaction had been significant and had stayed in the model, that something would have been associated with disease. For example, if the `disease:Species` table had looked like this:

```
disease = c("a", "a", "p", "p")
Species = c("A", "B", "A", "B")
frequency = c(10, 50, 30, 30)
xx = data.frame(disease, Species, frequency)
xtabs(frequency ~ disease + Species)
```

```
##        Species
## disease  A  B
##       a 10 50
##       p 30 30
```

For species A, disease is present 75% of the time, but for species B it's present less than 40% of the time. So in this one there ought to be a significant association between disease and species:

```
xx.1 = glm(frequency ~ disease * Species, data = xx,
    family = "poisson")
drop1(xx.1, test = "Chisq")
```

```
## Single term deletions
##
## Model:
## frequency ~ disease * Species
##                 Df Deviance    AIC    LRT
## <none>               0.000 28.400
## disease:Species  1   15.518 41.918 15.518
##                  Pr(>Chi)
## <none>
## disease:Species 8.171e-05 ***
## ---
## Signif. codes:
##   0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1  ' ' 1
```

And so there is. Nothing can come out of the model. (This is the same kind of test as a chi-squared test for association, if you know about that. The log-linear model is a multi-variable generalization of that.)

12.7 *Mating songs in crickets*

Male tree crickets produce "mating songs" by rubbing their wings together to produce a chirping sound. It is hypothesized that female tree crickets identify males of the correct species by how fast (in chirps per second) the male's mating song is. This is called the "pulse rate". Some data for two species of crickets are in link. The

columns, which are unlabelled, are temperature and pulse rate (respectively) for *Oecanthus exclamationis* (first two columns) and *Oecanthus niveus* (third and fourth columns). The columns are separated by tabs. There are some missing values in the first two columns because fewer *exclamationis* crickets than *niveus* crickets were measured. The research question is whether males of the different species have different average pulse rates. It is also of interest to see whether temperature has an effect, and if so, what. Before we get to that, however, we have some data organization to do.

(a) Read in the data, allowing for the fact that you have no column names. You'll see that the columns have names `X1` through `X4`. This is OK.

Solution

Tab-separated, so `read_tsv`; no column names, so `col_names=F`:

```
my_url = "http://www.utsc.utoronto.ca/~butler/c32/crickets.txt"
crickets = read_tsv(my_url, col_names = F)
```

```
## Parsed with column specification:
## cols(
##   X1 = col_double(),
##   X2 = col_double(),
##   X3 = col_double(),
##   X4 = col_double()
## )
```

```
crickets
```

```
## # A tibble: 17 x 4
##       X1    X2    X3    X4
##    <dbl> <dbl> <dbl> <dbl>
##  1  20.8  67.9  17.2  44.3
##  2  20.8  65.1  18.3  47.2
##  3  24    77.3  18.3  47.6
##  4  24    78.7  18.3  49.6
##  5  24    79.4  18.9  50.3
##  6  24    80.4  18.9  51.8
##  7  26.2  85.8  20.4  60
##  8  26.2  86.6  21    58.5
##  9  26.2  87.5  21    58.9
## 10  26.2  89.1  22.1  60.7
## 11  28.4  98.6  23.5  69.8
## 12  29   101.   24.2  70.9
## 13  30.4  99.3  25.9  76.2
```

```
## 14  30.4 102.   26.5  76.1
## 15  NA    NA    26.5  77
## 16  NA    NA    26.5  77.7
## 17  NA    NA    28.6  84.7
```

As promised.

If you didn't catch the tab-separated part, this probably happened to you:

```
d = read_delim(my_url, " ", col_names = F)
```

```
## Parsed with column specification:
## cols(
##   X1 = col_character()
## )
```

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

```
## Warning: 3 parsing failures.
## row # A tibble: 3 x 5 col     row col   expected  actual file          expected   <int> <chr> <chr>
```

This doesn't look good:

```
problems(d)
```

```
## # A tibble: 3 x 5
##     row col   expected  actual file
##   <int> <chr> <chr>     <chr>  <chr>
## 1    15 <NA>  1 columns 2 col~ 'http://www.~
## 2    16 <NA>  1 columns 2 col~ 'http://www.~
## 3    17 <NA>  1 columns 2 col~ 'http://www.~
```

The "expected columns" being 1 should bother you, since we know there are supposed to be 4 columns. At this point, we take a look at what got read in:

```
d
```

```
## # A tibble: 17 x 1
##    X1
##    <chr>
##  1 "20.8\t67.9\t17.2\t44.3"
##  2 "20.8\t65.1\t18.3\t47.2"
##  3 "24.0\t77.3\t18.3\t47.6"
##  4 "24.0\t78.7\t18.3\t49.6"
##  5 "24.0\t79.4\t18.9\t50.3"
```

```
##  6 "24.0\t80.4\t18.9\t51.8"
##  7 "26.2\t85.8\t20.4\t60.0"
##  8 "26.2\t86.6\t21.0\t58.5"
##  9 "26.2\t87.5\t21.0\t58.9"
## 10 "26.2\t89.1\t22.1\t60.7"
## 11 "28.4\t98.6\t23.5\t69.8"
## 12 "29.0\t100.8\t24.2\t70.9"
## 13 "30.4\t99.3\t25.9\t76.2"
## 14 "30.4\t101.7\t26.5\t76.1"
## 15 "NA\tNA"
## 16 "NA\tNA"
## 17 "NA\tNA"
```

and there you see the `t` or "tab" characters separating the values, instead of spaces. (This is what I tried first, and once I looked at this, I realized that `read_tsv` was what I needed.)

(b) These data are rather far from being tidy. There need to be three variables, temperature, pulse rate and species, and there are $14 + 17 = 31$ observations altogether. This one is tricky in that there are temperature and pulse rate for each of two levels of a factor, so I'll suggest combining the temperature and chirp rate together into one thing for each species, then gathering them, then splitting them again. Create new columns, named for each species, that contain the temperature and pulse rate for that species in that order, `united` together. For the rest of this question, start from the data frame you read in, and build a pipe, one or two steps at a time, to save creating a lot of temporary data frames.

Solution

Breathe, and then begin. `unite` creates new columns by joining together old ones:

As *str-c* or *paste* do, actually, but the advantage of *unite* is that it gets rid of the other columns, which you probably no longer need.

```
crickets %>% unite(exclamationis, X1:X2) %>% unite(niveus,
    X3:X4)
```

```
## # A tibble: 17 x 2
##    exclamationis niveus
##    <chr>         <chr>
##  1 20.8_67.9     17.2_44.3
##  2 20.8_65.1     18.3_47.2
##  3 24_77.3       18.3_47.6
##  4 24_78.7       18.3_49.6
##  5 24_79.4       18.9_50.3
##  6 24_80.4       18.9_51.8
##  7 26.2_85.8     20.4_60
```

```
##  8 26.2_86.6     21_58.5
##  9 26.2_87.5     21_58.9
## 10 26.2_89.1     22.1_60.7
## 11 28.4_98.6     23.5_69.8
## 12 29_100.8      24.2_70.9
## 13 30.4_99.3     25.9_76.2
## 14 30.4_101.7    26.5_76.1
## 15 NA_NA         26.5_77
## 16 NA_NA         26.5_77.7
## 17 NA_NA         28.6_84.7
```

Note that the original columns `X1:X4` are *gone*, which is fine, because the information we needed from them is contained in the two new columns. `unite` by default uses an underscore to separate the joined-together values, which is generally safe since you won't often find those in data.

Digression: `unite`-ing with a space could cause problems if the data values have spaces in them already. Consider this list of names:

```
names = c("Cameron McDonald", "Durwin Yang", "Ole Gunnar Solskjaer",
    "Mahmudullah")
```

Two very former students of mine, a Norwegian soccer player, and a Bangladeshi cricketer. Only one of these has played for Manchester United:

```
manu = c(F, F, T, F)
```

and let's make a data frame:

```
d = tibble(name = names, manu = manu)
d
```

```
## # A tibble: 4 x 2
##   name                 manu
##   <chr>                <lgl>
## 1 Cameron McDonald     FALSE
## 2 Durwin Yang          FALSE
## 3 Ole Gunnar Solskjaer TRUE
## 4 Mahmudullah          FALSE
```

Now, what happens if we `unite` those columns, separating them by a space?

```
d %>% unite(joined, name:manu, sep = " ")
```

```
## # A tibble: 4 x 1
```

```
##   joined
##   <chr>
## 1 Cameron McDonald FALSE
## 2 Durwin Yang FALSE
## 3 Ole Gunnar Solskjaer TRUE
## 4 Mahmudullah FALSE
```

If we then try to separate them again, what happens?

```
d %>% unite(joined, name:manu, sep = " ") %>%
    separate(joined, c("one", "two"), " ")
```

```
## Warning: Expected 2 pieces. Additional pieces
## discarded in 3 rows [1, 2, 3].
```

```
## # A tibble: 4 x 2
##   one         two
##   <chr>       <chr>
## 1 Cameron     McDonald
## 2 Durwin      Yang
## 3 Ole         Gunnar
## 4 Mahmudullah FALSE
```

Things have gotten lost: most of the original values of `manu` and some of the names. If we use a different separator character, either choosing one deliberately or going with the default underscore, everything works swimmingly:

```
d %>% unite(joined, name:manu, sep = ":") %>%
    separate(joined, c("one", "two"), ":")
```

```
## # A tibble: 4 x 2
##   one                  two
##   <chr>                <chr>
## 1 Cameron McDonald     FALSE
## 2 Durwin Yang          FALSE
## 3 Ole Gunnar Solskjaer TRUE
## 4 Mahmudullah          FALSE
```

and we are back to where we started.

If you run just the `unite` line (move the pipe symbol to the next line so that the `unite` line is complete as it stands), you'll see what happened.

(c) The two columns `exclamationis` and `niveus` that you just created are both temperature-pulse rate combos, but for different species. `gather` them together into one column, labelled by species. (This is a straight `tidyr gather`, even though they contain something odd-looking.)

Solution

Thus, this, naming the new column `temp_pulse` since it contains both of those things. Add to the end of the pipe you started building in the previous part:

```
crickets %>% unite(exclamationis, X1:X2) %>% unite(niveus,
    X3:X4) %>% gather(species, temp_pulse, exclamationis:niveus)
```

```
## # A tibble: 34 x 2
##    species       temp_pulse
##    <chr>         <chr>
##  1 exclamationis 20.8_67.9
##  2 exclamationis 20.8_65.1
##  3 exclamationis 24_77.3
##  4 exclamationis 24_78.7
##  5 exclamationis 24_79.4
##  6 exclamationis 24_80.4
##  7 exclamationis 26.2_85.8
##  8 exclamationis 26.2_86.6
##  9 exclamationis 26.2_87.5
## 10 exclamationis 26.2_89.1
## # ... with 24 more rows
```

Yep. If you scroll down with Next, you'll see the other species of crickets, and you'll see some missing values at the bottom, labelled, at the moment, `NA_NA`.

This is going to get rather long, but don't fret: we debugged the two `unite` lines before, so if you get any errors, they must have come from the `gather`. So that would be the place to check.

(d) Now split up the temperature-pulse combos at the underscore, into two separate columns. This is `separate`. When specifying what to separate by, you can use a number ("split after this many characters") or a piece of text, in quotes ("when you see this text, split at it").

Solution

The text to split by is an underscore (in quotes), since `unite` by default puts an underscore in between the values it pastes together. Glue the `separate` onto the end. We are creating two new variables `temperature` and `pulse_rate`:

```
crickets %>% unite(exclamationis, X1:X2) %>% unite(niveus,
    X3:X4) %>% gather(species, temp_pulse, exclamationis:niveus) %>%
    separate(temp_pulse, c("temperature", "pulse_rate"),
        "_")
```

```
## # A tibble: 34 x 3
##    species       temperature pulse_rate
##    <chr>         <chr>       <chr>
##  1 exclamationis 20.8        67.9
##  2 exclamationis 20.8        65.1
##  3 exclamationis 24          77.3
##  4 exclamationis 24          78.7
##  5 exclamationis 24          79.4
##  6 exclamationis 24          80.4
##  7 exclamationis 26.2        85.8
##  8 exclamationis 26.2        86.6
##  9 exclamationis 26.2        87.5
## 10 exclamationis 26.2        89.1
## # ... with 24 more rows
```

You'll note that `unite` and `separate` are opposites ("inverses") of each other, but we haven't just done something and then undone it, because we have a `gather` in between; in fact, arranging it this way has done precisely the tidying we wanted.

(e) Almost there. Temperature and pulse rate are still text (because `unite` turned them into text), but they should be numbers. Create new variables that are numerical versions of temperature and pulse rate (using `as.numeric`). Check that you have no extraneous variables (and, if necessary, get rid of the ones you don't want). (Species is also text and really ought to be a factor, but having it as text doesn't seem to cause any problems.) You can, if you like, use `parse_number` instead of `as.numeric`. They should both work. The distinction I prefer to make is that `parse_number` is good for text with a number in it (that we want to pull the number out of), while `as.numeric` is for turning something that looks like a number but isn't one into a genuine number.

You could just as well make the point that the text 20.8 contains the number 20.8 and nothing else, so that parsing it as text in search of a number will pull out 20.8 as a number. If that logic works for you, go with it.

Solution

`mutate`-ing into a column that already exists overwrites the variable that's already there (which saves us some effort here).

```
crickets.1 = crickets %>% unite(exclamationis,
    X1:X2) %>% unite(niveus, X3:X4) %>% gather(species,
    temp_pulse, exclamationis:niveus) %>% separate(temp_pulse,
    c("temperature", "pulse_rate"), "_") %>% mutate(temperature = as.numeric(temperature)) %>%
    mutate(pulse_rate = as.numeric(pulse_rate))
```

```
## Warning in evalq(as.numeric(temperature),
## <environment>): NAs introduced by coercion
```

```
## Warning in evalq(as.numeric(pulse_rate),
## <environment>): NAs introduced by coercion
```

```
crickets.1
```

```
## # A tibble: 34 x 3
##    species       temperature pulse_rate
##    <chr>               <dbl>      <dbl>
##  1 exclamationis        20.8       67.9
##  2 exclamationis        20.8       65.1
##  3 exclamationis        24         77.3
##  4 exclamationis        24         78.7
##  5 exclamationis        24         79.4
##  6 exclamationis        24         80.4
##  7 exclamationis        26.2       85.8
##  8 exclamationis        26.2       86.6
##  9 exclamationis        26.2       87.5
## 10 exclamationis        26.2       89.1
## # ... with 24 more rows
```

I saved the data frame this time, since this is the one we will use for our analysis.

The warning message tells us that we got genuine missing-value NAs back, which is probably what we want. Specifically, they got turned from missing *text* to missing *numbers*! The R word "coercion" means values being changed from one type of thing to another type of thing. (We'll ignore the missings and see if they cause us any trouble. The same warning messages will show up on graphs later.) So I have 34 rows (including three rows of missings) instead of the 31 rows I would have liked. Otherwise, success.

You might think that missing is just missing, but R distinguishes between types of missing.

There is (inevitably) another way to do this. We are doing the `as.numeric` twice, exactly the same on two different columns, and when you are doing the same thing on a number of columns, here a `mutate` with the same function, you have the option of using `mutate_if` or `mutate_at`. These are like `summarize_if` and `summarize_at` that we used way back to compute numerical summaries of a bunch of columns: the `if` variant works on columns that share a property, like being numeric, and the `at` variant works on columns whose names have something in common or that we can list, which is what we want here:

```
crickets %>% unite(exclamationis, X1:X2) %>% unite(niveus,
    X3:X4) %>% gather(species, temp_pulse, exclamationis:niveus) %>%
    separate(temp_pulse, c("temperature", "pulse_rate"),
        "_") %>% mutate_at(vars(temperature:pulse_rate),
    funs(as.numeric))
```

```
## Warning in evalq(as.numeric(temperature),
## <environment>): NAs introduced by coercion

## Warning in evalq(as.numeric(pulse_rate),
## <environment>): NAs introduced by coercion

## # A tibble: 34 x 3
##    species       temperature pulse_rate
##    <chr>               <dbl>      <dbl>
##  1 exclamationis        20.8       67.9
##  2 exclamationis        20.8       65.1
##  3 exclamationis        24         77.3
##  4 exclamationis        24         78.7
##  5 exclamationis        24         79.4
##  6 exclamationis        24         80.4
##  7 exclamationis        26.2       85.8
##  8 exclamationis        26.2       86.6
##  9 exclamationis        26.2       87.5
## 10 exclamationis        26.2       89.1
## # ... with 24 more rows
```

Can't I just say that these are columns 2 and 3?

```
crickets %>% unite(exclamationis, X1:X2) %>% unite(niveus,
    X3:X4) %>% gather(species, temp_pulse, exclamationis:niveus) %>%
    separate(temp_pulse, c("temperature", "pulse_rate"),
        "_") %>% mutate_at(vars(2:3), funs(as.numeric))
```

```
## Warning in evalq(as.numeric(temperature),
## <environment>): NAs introduced by coercion

## Warning in evalq(as.numeric(pulse_rate),
## <environment>): NAs introduced by coercion

## # A tibble: 34 x 3
##    species       temperature pulse_rate
##    <chr>               <dbl>      <dbl>
##  1 exclamationis        20.8       67.9
##  2 exclamationis        20.8       65.1
##  3 exclamationis        24         77.3
##  4 exclamationis        24         78.7
##  5 exclamationis        24         79.4
##  6 exclamationis        24         80.4
##  7 exclamationis        26.2       85.8
##  8 exclamationis        26.2       86.6
##  9 exclamationis        26.2       87.5
## 10 exclamationis        26.2       89.1
## # ... with 24 more rows
```

Yes. Equally good. What goes into the `vars` is the same as can go into a `select`: column numbers, names, or any of those "select helpers" like `starts_with`.

You might think of `mutate_if` here, but if you scroll back, you'll find that all the columns are text, before you convert temperature and pulse rate to numbers, and so there's no way to pick out just the two columns you want that way.

Check that the temperature and pulse rate columns are now labelled `dbl`, which means they actually *are* decimal numbers (and don't just look like decimal numbers).

Either way, using `unite` and then `separate` means that all the columns we created we want to keep (or, all the ones we would have wanted to get rid of have already been gotten rid of).

Now we could actually do some statistics. That we do elsewhere.

12.8 Cars

My cars data file can be found at link. The values in the data file are separated by commas; the car names are up to 29 characters long. Display your results for each part after (a). In R, displaying a `tibble` normally shows its first ten lines, which is all you need here; there's no need to display all the lines.

(a) Read the data into R and list the values.

Solution

`read_csv` will do it:

```
my_url = "http://www.utsc.utoronto.ca/~butler/c32/cars.csv"
cars = read_csv(my_url)
```

```
## Parsed with column specification:
## cols(
##   car = col_character(),
##   MPG = col_double(),
##   weight = col_double(),
##   cylinders = col_integer(),
##   hp = col_integer(),
##   country = col_character()
## )
```

```
cars
```

```
## # A tibble: 38 x 6
##    car    MPG weight cylinders    hp country
##    <ch> <dbl>  <dbl>     <int> <int> <chr>
```

```
##  1 Bui~  28.4   2.67         4    90 U.S.
##  2 Dod~  30.9   2.23         4    75 U.S.
##  3 Mer~  20.8   3.07         6    85 U.S.
##  4 Fia~  37.3   2.13         4    69 Italy
##  5 Peu~  16.2   3.41         6   133 France
##  6 VW ~  31.9   1.92         4    71 Germany
##  7 Ply~  34.2   2.2          4    70 U.S.
##  8 Maz~  34.1   1.98         4    65 Japan
##  9 Bui~  16.9   4.36         8   155 U.S.
## 10 Aud~  20.3   2.83         5   103 Germany
## # ... with 28 more rows
```

(b) Display only the car names and the countries they come from.

Solution

```
cars %>% select(car, country)
```

```
## # A tibble: 38 x 2
##    car                country
##    <chr>              <chr>
##  1 Buick Skylark      U.S.
##  2 Dodge Omni         U.S.
##  3 Mercury Zephyr     U.S.
##  4 Fiat Strada        Italy
##  5 Peugeot 694 SL     France
##  6 VW Rabbit          Germany
##  7 Plymouth Horizon   U.S.
##  8 Mazda GLC          Japan
##  9 Buick Estate Wagon U.S.
## 10 Audi 5000          Germany
## # ... with 28 more rows
```

This *almost* works, but not quite:

```
cars %>% select(starts_with("c"))
```

```
## # A tibble: 38 x 3
##    car                cylinders country
##    <chr>                  <int> <chr>
##  1 Buick Skylark              4 U.S.
##  2 Dodge Omni                 4 U.S.
##  3 Mercury Zephyr             6 U.S.
##  4 Fiat Strada                4 Italy
##  5 Peugeot 694 SL             6 France
##  6 VW Rabbit                  4 Germany
```

```
##  7 Plymouth Horizon           4 U.S.
##  8 Mazda GLC                  4 Japan
##  9 Buick Estate Wagon         8 U.S.
## 10 Audi 5000                  5 Germany
## # ... with 28 more rows
```

It gets *all* the columns that start with `c`, which includes `cylinders` as well.

(c) Display everything *except* horsepower:

Solution
Naming what you *don't* want is sometimes easier:

```
cars %>% select(-hp)
```

```
## # A tibble: 38 x 5
##    car          MPG weight cylinders country
##    <chr>      <dbl>  <dbl>     <int> <chr>
##  1 Buick Sky~  28.4   2.67         4 U.S.
##  2 Dodge Omni  30.9   2.23         4 U.S.
##  3 Mercury Z~  20.8   3.07         6 U.S.
##  4 Fiat Stra~  37.3   2.13         4 Italy
##  5 Peugeot 6~  16.2   3.41         6 France
##  6 VW Rabbit   31.9   1.92         4 Germany
##  7 Plymouth ~  34.2   2.2          4 U.S.
##  8 Mazda GLC   34.1   1.98         4 Japan
##  9 Buick Est~  16.9   4.36         8 U.S.
## 10 Audi 5000   20.3   2.83         5 Germany
## # ... with 28 more rows
```

(d) Display only the cars that have 8-cylinder engines (but display all the variables for those cars).

Solution
This:

```
cars %>% filter(cylinders == 8)
```

```
## # A tibble: 8 x 6
##   car     MPG weight cylinders    hp country
##   <chr> <dbl>  <dbl>     <int> <int> <chr>
## 1 Buic~  16.9   4.36         8   155 U.S.
## 2 Chev~  19.2   3.60         8   125 U.S.
## 3 Chry~  18.5   3.94         8   150 U.S.
## 4 Ford~  17.6   3.72         8   129 U.S.
## 5 Dodg~  18.2   3.83         8   135 U.S.
```

```
## 6 Ford~  15.5   4.05         8   142 U.S.
## 7 Merc~  16.5   3.96         8   138 U.S.
## 8 Chev~  17     3.84         8   130 U.S.
```

8 of them, all from the US.

(e) Display the cylinders and horsepower for the cars that have horsepower 70 or less.

Solution

This one is selecting some observations and some variables:

```
cars %>% filter(hp <= 70) %>% select(cylinders:hp)
```

```
## # A tibble: 6 x 2
##   cylinders    hp
##       <int> <int>
## 1         4    69
## 2         4    70
## 3         4    65
## 4         4    65
## 5         4    68
## 6         4    68
```

Cylinders and horsepower are consecutive columns, so we can select them either using the colon `:` or by `c(cylinders,hp)`.

You can also do the `filter` and the `select` the other way around. This one works because the *rows* you want to choose are determined by a column you're going to keep. If you wanted to display the cylinders and horsepower of the cars with `mpg` over 30, you would have to choose the rows first, because after you've chosen the columns, there is no `mpg` any more.

(f) Find the mean and SD of gas mileage of the cars with 4 cylinders.

Solution

```
cars %>% filter(cylinders == 4) %>% summarize(m = mean(MPG),
    s = sd(MPG))
```

```
## # A tibble: 1 x 2
##       m     s
##   <dbl> <dbl>
## 1  30.0  4.18
```

Or you can get the mean and SD of gas mileage for all numbers of cylinders, and pick out the one you want:

```
cars %>% group_by(cylinders) %>% summarize(m = mean(MPG),
    s = sd(MPG))
```

```
## # A tibble: 4 x 3
##   cylinders     m     s
##       <int> <dbl> <dbl>
## 1         4  30.0  4.18
## 2         5  20.3 NA
## 3         6  21.1  4.08
## 4         8  17.4  1.19
```

Top row is the same as before. And since the output is a data frame, you can do any of these things with *that*, for example:

```
cars %>% group_by(cylinders) %>% summarize(m = mean(MPG),
    s = sd(MPG)) %>% filter(cylinders == 4)
```

```
## # A tibble: 1 x 3
##   cylinders     m     s
##       <int> <dbl> <dbl>
## 1         4  30.0  4.18
```

to pick out just the right row. This is a very easy kind of question to set on an exam. Just so you know.

12.9 *Number 1 songs*

The data file link contains a lot of information about songs popular in 2000. This dataset is untidy. Our ultimate aim is to answer "which song occupied the #1 position for the largest number of weeks?". To do that, we will build a pipe that starts from the data frame read in from the URL above, and finishes with an answer to the question. I will take you through this step by step. Each part will involve adding something to the pipe you built previously (possibly after removing a line or two that you used to display the previous result).

(a) Read the data and display what you have.

Solution

```
billboard = read_csv("http://stat405.had.co.nz/data/billboard.csv")
```

```
## Parsed with column specification:
## cols(
##   .default = col_integer(),
##   artist.inverted = col_character(),
##   track = col_character(),
```

```
##   time = col_time(format = ""),
##   genre = col_character(),
##   date.entered = col_date(format = ""),
##   date.peaked = col_date(format = ""),
##   x66th.week = col_character(),
##   x67th.week = col_character(),
##   x68th.week = col_character(),
##   x69th.week = col_character(),
##   x70th.week = col_character(),
##   x71st.week = col_character(),
##   x72nd.week = col_character(),
##   x73rd.week = col_character(),
##   x74th.week = col_character(),
##   x75th.week = col_character(),
##   x76th.week = col_character()
## )

## See spec(...) for full column specifications.
```

There are a *lot* of columns. What does this look like?

```
billboard
```

```
## # A tibble: 317 x 83
##     year artist.inverted track time  genre
##    <int> <chr>           <chr> <tim> <chr>
##  1  2000 Destiny's Child Inde~ 03:38 Rock
##  2  2000 Santana         Mari~ 04:18 Rock
##  3  2000 Savage Garden   I Kn~ 04:07 Rock
##  4  2000 Madonna         Music 03:45 Rock
##  5  2000 Aguilera, Chri~ Come~ 03:38 Rock
##  6  2000 Janet           Does~ 04:17 Rock
##  7  2000 Destiny's Child Say ~ 04:31 Rock
##  8  2000 Iglesias, Enri~ Be W~ 03:36 Latin
##  9  2000 Sisqo           Inco~ 03:52 Rock
## 10  2000 Lonestar        Amaz~ 04:25 Coun~
## # ... with 307 more rows, and 78 more
## #   variables: date.entered <date>,
## #   date.peaked <date>, x1st.week <int>,
## #   x2nd.week <int>, x3rd.week <int>,
## #   x4th.week <int>, x5th.week <int>,
## #   x6th.week <int>, x7th.week <int>,
## #   x8th.week <int>, x9th.week <int>,
## #   x10th.week <int>, x11th.week <int>,
## #   x12th.week <int>, x13th.week <int>,
## #   x14th.week <int>, x15th.week <int>,
```

```
## #   x16th.week <int>, x17th.week <int>,
## #   x18th.week <int>, x19th.week <int>,
## #   x20th.week <int>, x21st.week <int>,
## #   x22nd.week <int>, x23rd.week <int>,
## #   x24th.week <int>, x25th.week <int>,
## #   x26th.week <int>, x27th.week <int>,
## #   x28th.week <int>, x29th.week <int>,
## #   x30th.week <int>, x31st.week <int>,
## #   x32nd.week <int>, x33rd.week <int>,
## #   x34th.week <int>, x35th.week <int>,
## #   x36th.week <int>, x37th.week <int>,
## #   x38th.week <int>, x39th.week <int>,
## #   x40th.week <int>, x41st.week <int>,
## #   x42nd.week <int>, x43rd.week <int>,
## #   x44th.week <int>, x45th.week <int>,
## #   x46th.week <int>, x47th.week <int>,
## #   x48th.week <int>, x49th.week <int>,
## #   x50th.week <int>, x51st.week <int>,
## #   x52nd.week <int>, x53rd.week <int>,
## #   x54th.week <int>, x55th.week <int>,
## #   x56th.week <int>, x57th.week <int>,
## #   x58th.week <int>, x59th.week <int>,
## #   x60th.week <int>, x61st.week <int>,
## #   x62nd.week <int>, x63rd.week <int>,
## #   x64th.week <int>, x65th.week <int>,
## #   x66th.week <chr>, x67th.week <chr>,
## #   x68th.week <chr>, x69th.week <chr>,
## #   x70th.week <chr>, x71st.week <chr>,
## #   x72nd.week <chr>, x73rd.week <chr>,
## #   x74th.week <chr>, x75th.week <chr>,
## #   x76th.week <chr>
```

On yours, you will definitely see a little arrow top right saying "there are more columns", and you will have to click on it several times to see them all.

(b) The columns `x1st.week` through `x76th.week` contain the rank of each song in the Billboard chart in that week, with week 1 being the first week that the song appeared in the chart. Convert all these columns into two: an indication of week, called `week`, and of rank, called `rank`. Most songs appeared in the Billboard chart for a lot less than 76 weeks, so there are missing values, which you want to remove. (I say "indication of week" since this will probably be text at the moment). Display your new data frame. Do you have fewer columns? Why do you have a lot more rows?

Explain briefly.

Solution

This is `gathering` up all those columns, with `na.rm=T` to get rid of the missings:

```
billboard %>% gather(week, rank, x1st.week:x76th.week,
    na.rm = T)
```

```
## # A tibble: 5,307 x 9
##     year artist.inverted track time  genre
##   * <int> <chr>           <chr> <tim> <chr>
##  1  2000 Destiny's Child Inde~ 03:38 Rock
##  2  2000 Santana         Mari~ 04:18 Rock
##  3  2000 Savage Garden   I Kn~ 04:07 Rock
##  4  2000 Madonna         Music 03:45 Rock
##  5  2000 Aguilera, Chri~ Come~ 03:38 Rock
##  6  2000 Janet           Does~ 04:17 Rock
##  7  2000 Destiny's Child Say ~ 04:31 Rock
##  8  2000 Iglesias, Enri~ Be W~ 03:36 Latin
##  9  2000 Sisqo           Inco~ 03:52 Rock
## 10  2000 Lonestar        Amaz~ 04:25 Coun~
## # ... with 5,297 more rows, and 4 more
## #   variables: date.entered <date>,
## #   date.peaked <date>, week <chr>,
## #   rank <chr>
```

Another way to do this is with a select-helper: all those column names end with `week`, so we can select them all thus:

```
billboard %>% gather(week, rank, ends_with("week"),
    na.rm = T)
```

```
## # A tibble: 5,307 x 9
##     year artist.inverted track time  genre
##   * <int> <chr>           <chr> <tim> <chr>
##  1  2000 Destiny's Child Inde~ 03:38 Rock
##  2  2000 Santana         Mari~ 04:18 Rock
##  3  2000 Savage Garden   I Kn~ 04:07 Rock
##  4  2000 Madonna         Music 03:45 Rock
##  5  2000 Aguilera, Chri~ Come~ 03:38 Rock
##  6  2000 Janet           Does~ 04:17 Rock
##  7  2000 Destiny's Child Say ~ 04:31 Rock
##  8  2000 Iglesias, Enri~ Be W~ 03:36 Latin
##  9  2000 Sisqo           Inco~ 03:52 Rock
## 10  2000 Lonestar        Amaz~ 04:25 Coun~
```

```
## # ... with 5,297 more rows, and 4 more
## #   variables: date.entered <date>,
## #   date.peaked <date>, week <chr>,
## #   rank <chr>
```

There are now only 9 columns, a lot fewer than we started with. This is (I didn't need you to say) because we have collected together all those `week` columns into one (a column called `rank` with an indication of which `week` it came from). The logic of the `gather` is that all those columns contain ranks (which is what make them the same), but they are ranks from different weeks (which is what makes them different).

What has actually happened is that we have turned "wide" format into "long" format. This is not very insightful, so I would like you to go a bit further in your explanation. The original data frame encodes the rank of each song in each week, and what the `gather` has done is to make that explicit: in the new data frame, each song's rank in each week appears in *one* row, so that there are as many rows as there are song-week combinations. The original data frame had 317 songs over 76 weeks, so this many:

```
317 * 76
```

```
## [1] 24092
```

song-week combinations.

Not every song appeared in the Billboard chart for 76 weeks, so our tidy data frame has a lot fewer rows than this.

You need to say that the original data frame had each song appearing once (on one line), but now each song appears on multiple rows, one for each week that the song was in the chart. Or something equivalent to that.

(c) Display just your two new columns (for the first few rows). Add something appropriate onto the end of your pipe to do this.

Solution

A `select` is the thing:

```
billboard %>% gather(week, rank, x1st.week:x76th.week,
    na.rm = T) %>% select(week, rank)
```

```
## # A tibble: 5,307 x 2
##    week      rank
##  * <chr>     <chr>
##  1 x1st.week 78
##  2 x1st.week 15
```

```
##  3 x1st.week 71
##  4 x1st.week 41
##  5 x1st.week 57
##  6 x1st.week 59
##  7 x1st.week 83
##  8 x1st.week 63
##  9 x1st.week 77
## 10 x1st.week 81
## # ... with 5,297 more rows
```

(d) Both your `week` and `rank` columns are (probably) text. Create new columns that contain just the numeric values, and display just your new columns, again adding onto the end of your pipe. (In the previous part, you probably had some code that picked out a few columns to display them. Get rid of that.)

Solution

`parse_number` is the easiest. Create new columns, with `mutate`, that are the `parse_number`-ed versions of the old ones.

```
billboard %>% gather(week, rank, x1st.week:x76th.week,
    na.rm = T) %>% mutate(week_number = parse_number(week),
    rank_number = parse_number(rank)) %>% select(ends_with("number"))
```

```
## # A tibble: 5,307 x 2
##    week_number rank_number
##          <dbl>       <dbl>
##  1           1          78
##  2           1          15
##  3           1          71
##  4           1          41
##  5           1          57
##  6           1          59
##  7           1          83
##  8           1          63
##  9           1          77
## 10           1          81
## # ... with 5,297 more rows
```

You see that these are indeed numbers. (I gave my new columns names that ended with `number`, which meant that I could select them with the select-helper `ends_with`. I'm not insisting that you do this, but it's a handy trick.)

Since `rank` already looks like a number, but happens to be text, you can also convert it with `as.numeric` (or `as.integer`, since it is actually text that looks like a whole number). For converting `rank`,

these are also good, but for converting `week`, you need something that will pull out the number. (`str_extract` from `stringr` will also do it, but that's beyond our scope now. It's on page 212 of the R book if you wish to investigate it. But `parse_number` is a lot easier.)

(e) The meaning of your week-number column is that it refers to the number of weeks *after* the song first appeared in the Billboard chart. That is, if a song's first appearance (in `date.entered`) is July 24, then week 1 is July 24, week 2 is July 31, week 3 is August 7, and so on. Create a column `current` by adding the appropriate number of *days*, based on your week number, to `date.entered`. Display `date.entered`, your week number, and `current` to show that you have calculated the right thing. Note that you can add a number of days onto a date and you will get another date.

Solution

There is a (small) gotcha here: if you read carefully, you'll see that "week 1" is actually "week 0" in terms of the number of days to add on to `date.entered`. So you have to subtract one from the number of weeks before you multiply it by seven to get a number of days. After that thinking, this:

```
billboard %>% gather(week, rank, x1st.week:x76th.week,
    na.rm = T) %>% mutate(week_number = parse_number(week),
    rank_number = parse_number(rank)) %>% mutate(current = date.entered +
    (week_number - 1) * 7) %>% select(date.entered,
    week_number, current)
```

```
## # A tibble: 5,307 x 3
##    date.entered week_number current
##    <date>             <dbl> <date>
##  1 2000-09-23             1 2000-09-23
##  2 2000-02-12             1 2000-02-12
##  3 1999-10-23             1 1999-10-23
##  4 2000-08-12             1 2000-08-12
##  5 2000-08-05             1 2000-08-05
##  6 2000-06-17             1 2000-06-17
##  7 1999-12-25             1 1999-12-25
##  8 2000-04-01             1 2000-04-01
##  9 2000-06-24             1 2000-06-24
## 10 1999-06-05             1 1999-06-05
## # ... with 5,297 more rows
```

Don't forget to use your `week`-turned-into-number, or else it won't work! (This bit me too, so you don't need to feel bad.)

You can also combine the three column-definition statements into one mutate. It doesn't matter; as soon as you have defined a column, you can use it in defining another column, even within the same `mutate`.

Anyway, the rows displayed are all `week_number` 1, so the `current` date should be the same as `date.entered`, and is. (These are all the first week that a song is in the Billboard chart).

You might be thinking that this is not much of a check, and you would be right. A handy trick is to display a random sample of 10 (say) out of the 5,000-odd rows of the data frame. To do that, add the line `sample_n(10)` on the end, like this:

```
billboard %>% gather(week, rank, x1st.week:x76th.week,
    na.rm = T) %>% mutate(week_number = parse_number(week),
    rank_number = parse_number(rank)) %>% mutate(current = date.entered +
    (week_number - 1) * 7) %>% select(date.entered,
    week_number, current) %>% sample_n(10)
```

```
## # A tibble: 10 x 3
##    date.entered week_number current
##    <date>             <dbl> <date>
##  1 2000-03-18            27 2000-09-16
##  2 2000-08-12             3 2000-08-26
##  3 2000-09-23             4 2000-10-14
##  4 1999-12-04            11 2000-02-12
##  5 2000-01-22             8 2000-03-11
##  6 2000-05-27            16 2000-09-09
##  7 2000-04-29             5 2000-05-27
##  8 2000-06-10             4 2000-07-01
##  9 2000-04-15            12 2000-07-01
## 10 2000-08-26            18 2000-12-23
```

This gives a variety of rows to check. The first `current` should be $27 - 1 = 26$ weeks, or about 6 months, after the date the song entered the chart, and so it is; the second one should be $3 - 1 = 2$ weeks after entry, and it is. The third one should be 3 weeks after September 23; this is (as I figure it) September $23 + 21 = 44$, and September has 30 days, so this is really October 14. Check.

Your random selection of rows is likely to be different from mine, but the same kind of thinking will enable you to check whether it makes sense.

(f) Reaching the #1 rank on the Billboard chart is one of the highest accolades in the popular music world. List all the songs that reached `rank` 1. For these songs, list the artist (as given in the data set), the song title, and the date(s) for which the song was ranked

number 1. Arrange the songs in date order of being ranked #1. Display all the songs (I found 55 of them).

Solution

To the previous pipe, add the last lines below. You can use either `rank` (text) or what I called `rank_number` (a number). It doesn't matter here, since we are only checking for equal-to, not something like "less than":

```
billboard %>% gather(week, rank, x1st.week:x76th.week,
    na.rm = T) %>% mutate(week_number = parse_number(week),
    rank_number = parse_number(rank)) %>% mutate(current = date.entered +
    (week_number - 1) * 7) %>% filter(rank ==
    1) %>% arrange(current) %>% select(artist.inverted,
    track, current)
```

```
## # A tibble: 55 x 3
##    artist.inverted   track        current
##    <chr>             <chr>        <date>
##  1 Aguilera, Christ~ What A Girl~ 2000-01-15
##  2 Aguilera, Christ~ What A Girl~ 2000-01-22
##  3 Savage Garden     I Knew I Lo~ 2000-01-29
##  4 Savage Garden     I Knew I Lo~ 2000-02-05
##  5 Savage Garden     I Knew I Lo~ 2000-02-12
##  6 Carey, Mariah     Thank God I~ 2000-02-19
##  7 Savage Garden     I Knew I Lo~ 2000-02-26
##  8 Lonestar          Amazed       2000-03-04
##  9 Lonestar          Amazed       2000-03-11
## 10 Destiny's Child   Say My Name  2000-03-18
## # ... with 45 more rows
```

You'll see the first ten rows, as here, but with clickable buttons to see the next 10 (and the previous 10 if you have moved beyond 1–10). The "artist" column is called `artist.inverted` because, if the artist is a single person rather than a group, their last name is listed first. The song title appears in the column `track`.

The song by Destiny's Child spills into 2001 because it entered the chart in 2000, and the data set keeps a record of all such songs until they drop out of the chart. I'm not sure what happened to the song that was #1 on January 8, 2000; maybe it entered the chart in 1999 and so is not listed here.

Which was the title of a song by Prince.

(g) Use R to find out which song held the #1 rank for the largest number of weeks. For this, you can assume that the song titles are all unique (if it's the same song title, it's the same song), but the artists might not be (for example, Madonna might have had two

different songs reach the #1 rank). The information you need is in the output you obtained for the previous part, so it's a matter of adding some code to the end of that. The last mark was for displaying *only* the song that was ranked #1 for the largest number of weeks, or for otherwise making it easy to see which song it was.

Solution

This is a question of using `count`, but on the `track` title:

```
billboard %>% gather(week, rank, x1st.week:x76th.week,
    na.rm = T) %>% mutate(week_number = parse_number(week),
    rank_number = parse_number(rank)) %>% mutate(current = date.entered +
    (week_number - 1) * 7) %>% filter(rank ==
    1) %>% arrange(current) %>% select(artist.inverted,
    track, current) %>% count(track)
```

```
## # A tibble: 17 x 2
##    track                                   n
##    <chr>                               <int>
##  1 Amazed                                  2
##  2 Bent                                    1
##  3 Be With You                             3
##  4 Come On Over Baby (All I Want Is Y~     4
##  5 Doesn't Really Matter                   3
##  6 Everything You Want                     1
##  7 I Knew I Loved You                      4
##  8 Incomplete                              2
##  9 Independent Women Part I               11
## 10 It's Gonna Be Me                        2
## 11 Maria, Maria                           10
## 12 Music                                   4
## 13 Say My Name                             3
## 14 Thank God I Found You                   1
## 15 Try Again                               1
## 16 What A Girl Wants                       2
## 17 With Arms Wide Open                     1
```

Then you can scan down the `n` column, find that the biggest number is 11, and say: it's the song "Independent Women Part I" by Destiny's Child. This is 3 points (out of 4, when the question was to be handed in).

But, this is a data frame, so anything we can do to a data frame we can do to this, like listing out only the row(s) where `n` is equal to its maximum value:

```
billboard %>% gather(week, rank, x1st.week:x76th.week,
    na.rm = T) %>% mutate(week_number = parse_number(week),
```

```
    rank_number = parse_number(rank)) %>% mutate(current = date.entered +
    (week_number - 1) * 7) %>% filter(rank ==
    1) %>% arrange(current) %>% select(artist.inverted,
    track, current) %>% count(track) %>% filter(n ==
    max(n))
```

```
## # A tibble: 1 x 2
##   track                       n
##   <chr>                   <int>
## 1 Independent Women Part I    11
```

or arranging them in (most logically, descending) order by `n` to make it easier to pick out the top one:

```
billboard %>% gather(week, rank, x1st.week:x76th.week,
    na.rm = T) %>% mutate(week_number = parse_number(week),
    rank_number = parse_number(rank)) %>% mutate(current = date.entered +
    (week_number - 1) * 7) %>% filter(rank ==
    1) %>% arrange(current) %>% select(artist.inverted,
    track, current) %>% count(track) %>% arrange(desc(n))
```

```
## # A tibble: 17 x 2
##    track                                n
##    <chr>                            <int>
##  1 Independent Women Part I            11
##  2 Maria, Maria                        10
##  3 Come On Over Baby (All I Want Is Y~  4
##  4 I Knew I Loved You                   4
##  5 Music                                4
##  6 Be With You                          3
##  7 Doesn't Really Matter                3
##  8 Say My Name                          3
##  9 Amazed                               2
## 10 Incomplete                           2
## 11 It's Gonna Be Me                     2
## 12 What A Girl Wants                    2
## 13 Bent                                 1
## 14 Everything You Want                  1
## 15 Thank God I Found You                1
## 16 Try Again                            1
## 17 With Arms Wide Open                  1
```

Either of those will net you the 4th point.

If you want to be a little bit more careful, you can make an artist-track combination as below. This would catch occasions where the same song by two different artists made it to #1, or two different

songs that happened to have the same title did. It's not very likely that the same artist would record two *different* songs with the same title, though it is possible that the same song by the same artist could appear in the Billboard chart on two different occasions.

As, for example, when Prince died.

I think I want to create an artist-song combo fairly early in my pipe, and then display *that* later, something like this. This means replacing `track` by my `combo` later in the pipe, wherever it appears:

```
billboard %>% gather(week, rank, x1st.week:x76th.week,
    na.rm = T) %>% mutate(week_number = parse_number(week),
    rank_number = parse_number(rank)) %>% mutate(combo = paste(track,
    artist.inverted, sep = " by ")) %>% mutate(current = date.entered +
    (week_number - 1) * 7) %>% filter(rank ==
    1) %>% arrange(current) %>% select(combo,
    current) %>% count(combo) %>% arrange(desc(n))
```

```
## # A tibble: 17 x 2
##    combo                                  n
##    <chr>                              <int>
##  1 Independent Women Part I by Destin~    11
##  2 Maria, Maria by Santana                10
##  3 Come On Over Baby (All I Want Is Y~     4
##  4 I Knew I Loved You by Savage Garden     4
##  5 Music by Madonna                        4
##  6 Be With You by Iglesias, Enrique        3
##  7 Doesn't Really Matter by Janet          3
##  8 Say My Name by Destiny's Child          3
##  9 Amazed by Lonestar                      2
## 10 Incomplete by Sisqo                     2
## 11 It's Gonna Be Me by N'Sync              2
## 12 What A Girl Wants by Aguilera, Chr~     2
## 13 Bent by matchbox twenty                 1
## 14 Everything You Want by Vertical Ho~     1
## 15 Thank God I Found You by Carey, Ma~     1
## 16 Try Again by Aaliyah                    1
## 17 With Arms Wide Open by Creed            1
```

I don't think it makes any difference here, but it might in other years, or if you look over several years where you might get cover versions of the same song performed by different artists.

Zero-point bonus: how many of these artists have you heard of? How many have your parents heard of? (I followed popular music quite closely much earlier than this, in the early 1980s in the UK. I remember both Madonna and U2 when they *first* became famous. U2's first single was called "Fire" and it just scraped into the UK top 40. Things changed after that.)

12.10 *Bikes on College*

The City of Toronto collects all kinds of data on aspects of life in the city. See link. One collection of data is records of the number of cyclists on certain downtown streets. The data in link are a record of the cyclists on College Street on the block west from Huron to Spadina on September 24, 2010. In the spreadsheet, each row relates to one cyclist. The first column is the time the cyclist was observed (to the nearest 15 minutes). After that, there are four pairs of columns. The observer filled in (exactly) one X in each pair of columns, according to whether (i) the cyclist was male or female, (ii) was or was not wearing a helmet, (iii) was or was not carrying a passenger on the bike, (iv) was or was not riding on the sidewalk. We want to create a tidy data frame that has the time in each row, and has columns containing appropriate values, often `TRUE` or `FALSE`, for each of the four variables measured.

I will lead you through the process, which will involve developing a (long) pipe, one step at a time.

(a) Take a look at the spreadsheet (using Excel or similar: this may open when you click the link). Are there any obvious header rows? Is there any extra material before the data start? Explain briefly.

Solution

This is what I see (you should see something that looks like this):

	A	B	C	D	E	F	G	H	I
1	Ontario Traffic Inc - Bicycle Counts								
2									
3	Location:	College St from Spadina Ave to Huron St				Date:	09/24/2010		
4									
5	Direction:	WB				Weather:	Clear		
6									
7	Start Time	Gender		Wearing Helmet		Passenger On The Bike		Riding On The Sidewalk	
8	(every 15 min)	Male	Female	Yes	No	Yes	No	Yes	No
9	7:00	X			X		X		X
10		X			X		X		X
11		X			X		X		X
12		X			X		X		X
13		X			X		X		X
14	7:15	X		X			X		X
15			X	X			X		X
16		X		X			X		X
17			X	X			X	X	

There are really *two* rows of headers (the rows highlighted in yellow). The actual information that says what the column pair is about is in the first of those two rows, and the second row indicates which category of the information above this column refers to. This is not the usual way that the column headers encode what the columns are about: we are used to having *one* column `gender` that would take the values `female` or `male`, or a column `helmet` containing the values `yes` or `no`. (You might be sensing `gather` here, which may be one way of tackling this, but I lead you into another idea below.) There are also

six lines above the highlighted ones that contain background information about this study. (This is where I got the information about the date of the study and which block of which street it is about.) I am looking for two things: the apparent header line is actually two lines (the ones in yellow), and there are extra lines above that which are not data.

(b) Read the data into an R data frame. Read *without* headers, and instruct R how many lines to skip over using `skip=` and a suitable number. When this is working, display the first few lines of your data frame. Note that your columns have names `X1` through `X9`.

Solution

The actual data start on line 9, so we need to skip 8 lines. `col_names=F` is the way to say that we have no column names (not ones that we want to use, anyway). Just typing the name of the data frame will display "a few" (that is, 10) lines of it, so that you can check it for plausibleness:

```
my_url = "http://www.utsc.utoronto.ca/~butler/c32/bikes.csv"
bikes = read_csv(my_url, skip = 8, col_names = F)
```

```
## Parsed with column specification:
## cols(
##   X1 = col_time(format = ""),
##   X2 = col_character(),
##   X3 = col_character(),
##   X4 = col_character(),
##   X5 = col_character(),
##   X6 = col_character(),
##   X7 = col_character(),
##   X8 = col_character(),
##   X9 = col_character()
## )
```

```
bikes
```

```
## # A tibble: 1,958 x 9
##    X1     X2    X3    X4    X5    X6    X7
##    <time> <chr> <chr> <chr> <chr> <chr> <chr>
##  1 07:00  X     <NA>  <NA>  X     <NA>  X
##  2    NA  X     <NA>  <NA>  X     <NA>  X
##  3    NA  X     <NA>  <NA>  X     <NA>  X
##  4    NA  X     <NA>  <NA>  X     <NA>  X
##  5    NA  X     <NA>  <NA>  X     <NA>  X
##  6 07:15  X     <NA>  X     <NA>  <NA>  X
```

```
##  7    NA  <NA>  X     X     <NA>  <NA>  X
##  8    NA  X     <NA>  X     <NA>  <NA>  X
##  9    NA  <NA>  X     X     <NA>  <NA>  X
## 10    NA  X     <NA>  X     <NA>  <NA>  X
## # ... with 1,948 more rows, and 2 more
## #   variables: X8 <chr>, X9 <chr>
```

This seems to have worked: a column with times in it, and four pairs of columns, with exactly one of each pair having an X in it. The variable names `X1` through `X9` were generated by `read_csv`, as it does when you read in data with `col_names=F`. The times are correctly `times`, and the other columns are all text. The blank cells in the spreadsheet have appeared in our data frame as "missing" (`NA`). The notation `<NA>` means "missing text" (as opposed to a missing number, say).

The first line in our data frame contains the first 7:00 (am) cyclist, so it looks as if we skipped the right number of lines.

(c) What do you notice about the times in your first column? What do you think those "missing" times should be?

Solution

There are some times and some missing values. It seems a reasonable guess that the person recording the data only recorded a time when a new period of 15 minutes had begun, so that the missing times should be the same as the previous non-missing one: For example, the first five rows are cyclists observed at 7:00 am (or, at least, between 7:00 and 7:15). So they should be recorded as 7:00, and the ones in rows 7–10 should be recorded as 7:15, and so on.

(d) Find something from the `tidyverse` that will fill in those missing values with the right thing. Start a pipe from the data frame you read in, that updates the appropriate column with the filled-in times.

Oh, what a giveaway.

Solution

`fill` from `tidyr` fills in the missing times with the previous non-missing value. (This will mean finding the help for `fill` in R Studio or online.) I told you it was a giveaway. If you look in the help for `fill` via `?fill` (or if you Google `tidyr::fill`, which is the full name for "the `fill` that lives in `tidyr`"), you'll see that it requires up to two things (not including the data frame): a column to fill, and a direction to fill it (the default of "down" is exactly what we want). Thus:

```
bikes %>% fill(X1)
```

```
## # A tibble: 1,958 x 9
##    X1     X2    X3    X4    X5    X6    X7
##    <time> <chr> <chr> <chr> <chr> <chr> <chr>
##  1 07:00  X     <NA>  <NA>  X     <NA>  X
##  2 07:00  X     <NA>  <NA>  X     <NA>  X
##  3 07:00  X     <NA>  <NA>  X     <NA>  X
##  4 07:00  X     <NA>  <NA>  X     <NA>  X
##  5 07:00  X     <NA>  <NA>  X     <NA>  X
##  6 07:15  X     <NA>  X     <NA>  <NA>  X
##  7 07:15  <NA>  X     X     <NA>  <NA>  X
##  8 07:15  X     <NA>  X     <NA>  <NA>  X
##  9 07:15  <NA>  X     X     <NA>  <NA>  X
## 10 07:15  X     <NA>  X     <NA>  <NA>  X
## # ... with 1,948 more rows, and 2 more
## #   variables: X8 <chr>, X9 <chr>
```

Success!

We will probably want to rename `X1` to something like `time`, so let's do that now before we forget. There is a `rename` that does about what you'd expect:

```
bikes %>% fill(X1) %>% rename(Time = X1)
```

```
## # A tibble: 1,958 x 9
##    Time   X2    X3    X4    X5    X6    X7
##    <time> <chr> <chr> <chr> <chr> <chr> <chr>
##  1 07:00  X     <NA>  <NA>  X     <NA>  X
##  2 07:00  X     <NA>  <NA>  X     <NA>  X
##  3 07:00  X     <NA>  <NA>  X     <NA>  X
##  4 07:00  X     <NA>  <NA>  X     <NA>  X
##  5 07:00  X     <NA>  <NA>  X     <NA>  X
##  6 07:15  X     <NA>  X     <NA>  <NA>  X
##  7 07:15  <NA>  X     X     <NA>  <NA>  X
##  8 07:15  X     <NA>  X     <NA>  <NA>  X
##  9 07:15  <NA>  X     X     <NA>  <NA>  X
## 10 07:15  X     <NA>  X     <NA>  <NA>  X
## # ... with 1,948 more rows, and 2 more
## #   variables: X8 <chr>, X9 <chr>
```

The only thing I keep forgetting is that the syntax of `rename` is "new name equals old name". Sometimes I think it's the other way around, and then I wonder why it doesn't work.

I gave it a capital T so as not to confuse it with other things in R called `time`.

(e) R's `ifelse` function works like `=IF` in Excel. You use it to create values for a new variable, for example in a `mutate`. The first input

to it is a logical condition (something that is either true or false); the second is the value your new variable should take if the condition is true, and the third is the value of your new variable if the condition is false. Create a new column `gender` in your data frame that is "male" or "female" depending on the value of your `X2` column, using `mutate`. (You can assume that exactly one of the second and third columns has an `X` in it.) Add your code to the end of your pipe and display (the first 10 rows of) the result.

Solution

Under the assumption we are making, we only have to look at column `X2` and we ignore `X3` totally:

```
bikes %>% fill(X1) %>% rename(Time = X1) %>% mutate(gender = ifelse(X2 ==
    "X", "male", "female"))
```

```
## # A tibble: 1,958 x 10
##    Time   X2    X3    X4    X5    X6    X7
##    <time> <chr> <chr> <chr> <chr> <chr> <chr>
##  1 07:00  X     <NA>  <NA>  X     <NA>  X
##  2 07:00  X     <NA>  <NA>  X     <NA>  X
##  3 07:00  X     <NA>  <NA>  X     <NA>  X
##  4 07:00  X     <NA>  <NA>  X     <NA>  X
##  5 07:00  X     <NA>  <NA>  X     <NA>  X
##  6 07:15  X     <NA>  X     <NA>  <NA>  X
##  7 07:15  <NA>  X     X     <NA>  <NA>  X
##  8 07:15  X     <NA>  X     <NA>  <NA>  X
##  9 07:15  <NA>  X     X     <NA>  <NA>  X
## 10 07:15  X     <NA>  X     <NA>  <NA>  X
## # ... with 1,948 more rows, and 3 more
## #   variables: X8 <chr>, X9 <chr>,
## #   gender <chr>
```

Oh, that didn't work. The gender column is either `male` or missing; the two missing ones here should say `female`. What happened? Let's just look at our logical condition this time:

```
bikes %>% fill(X1) %>% rename(Time = X1) %>% mutate(isX = (X2 ==
    "X"))
```

```
## # A tibble: 1,958 x 10
##    Time   X2    X3    X4    X5    X6    X7
##    <time> <chr> <chr> <chr> <chr> <chr> <chr>
##  1 07:00  X     <NA>  <NA>  X     <NA>  X
##  2 07:00  X     <NA>  <NA>  X     <NA>  X
##  3 07:00  X     <NA>  <NA>  X     <NA>  X
```

```
##  4 07:00  X     <NA>  <NA>  X     <NA>  X
##  5 07:00  X     <NA>  <NA>  X     <NA>  X
##  6 07:15  X     <NA>  X     <NA>  <NA>  X
##  7 07:15  <NA>  X     X     <NA>  <NA>  X
##  8 07:15  X     <NA>  X     <NA>  <NA>  X
##  9 07:15  <NA>  X     X     <NA>  <NA>  X
## 10 07:15  X     <NA>  X     <NA>  <NA>  X
## # ... with 1,948 more rows, and 3 more
## #   variables: X8 <chr>, X9 <chr>, isX <lgl>
```

This is not true and false, it is true and missing. The idea is that if `X2` is missing, we don't (in general) know what its value is: it might even be `X`! So if `X2` is missing, any comparison of it with another value ought to be missing as well.

That's in general. Here, we know where those missing values came from: they were blank cells in the spreadsheet, so we actually have more information.

Perhaps a better way to go is to test whether `X2` is missing (in which case, it's a female cyclist). R has a function `is.na` which is `TRUE` if the thing inside it is missing and `FALSE` if the thing inside it has some non-missing value. In our case, it goes like this:

```
bikes %>% fill(X1) %>% rename(Time = X1) %>% mutate(gender = ifelse(is.na(X2),
    "female", "male"))
```

```
## # A tibble: 1,958 x 10
##    Time   X2    X3    X4    X5    X6    X7
##    <time> <chr> <chr> <chr> <chr> <chr> <chr>
##  1 07:00  X     <NA>  <NA>  X     <NA>  X
##  2 07:00  X     <NA>  <NA>  X     <NA>  X
##  3 07:00  X     <NA>  <NA>  X     <NA>  X
##  4 07:00  X     <NA>  <NA>  X     <NA>  X
##  5 07:00  X     <NA>  <NA>  X     <NA>  X
##  6 07:15  X     <NA>  X     <NA>  <NA>  X
##  7 07:15  <NA>  X     X     <NA>  <NA>  X
##  8 07:15  X     <NA>  X     <NA>  <NA>  X
##  9 07:15  <NA>  X     X     <NA>  <NA>  X
## 10 07:15  X     <NA>  X     <NA>  <NA>  X
## # ... with 1,948 more rows, and 3 more
## #   variables: X8 <chr>, X9 <chr>,
## #   gender <chr>
```

Or you can test `X3` for missingness: if missing, it's male, otherwise it's female. That also works.

This made an assumption that the person recording the X's actually *did* mark an X in exactly one of the columns. For example, the

columns could *both* be missing, or *both* have an X in them. This gives us more things to check, at least three. `ifelse` is good for something with only two alternatives, but when you have more, `case_when` is much better. Here's how that goes. Our strategy is to check for three things: (i) `X2` has an `X` and `X3` is missing; (ii) `X2` is missing and `X3` has an `X`; (iii) anything else, which is an error:

In some languages it is called *switch*. Python appears not to have it. What you do there instead is to use a Python dictionary to pick out the value you want.

```
bikes %>% fill(X1) %>% rename(Time = X1) %>% mutate(gender = case_when(X2 ==
    "X" & is.na(X3) ~ "Male", is.na(X2) & X3 ==
    "X" ~ "Female", TRUE ~ "Error!"))
```

```
## # A tibble: 1,958 x 10
##    Time   X2    X3    X4    X5    X6    X7
##    <time> <chr> <chr> <chr> <chr> <chr> <chr>
##  1 07:00  X     <NA>  <NA>  X     <NA>  X
##  2 07:00  X     <NA>  <NA>  X     <NA>  X
##  3 07:00  X     <NA>  <NA>  X     <NA>  X
##  4 07:00  X     <NA>  <NA>  X     <NA>  X
##  5 07:00  X     <NA>  <NA>  X     <NA>  X
##  6 07:15  X     <NA>  X     <NA>  <NA>  X
##  7 07:15  <NA>  X     X     <NA>  <NA>  X
##  8 07:15  X     <NA>  X     <NA>  <NA>  X
##  9 07:15  <NA>  X     X     <NA>  <NA>  X
## 10 07:15  X     <NA>  X     <NA>  <NA>  X
## # ... with 1,948 more rows, and 3 more
## #   variables: X8 <chr>, X9 <chr>,
## #   gender <chr>
```

It seems nicest to format it like that, with the squiggles lining up, so you can see what possible values `gender` might take.

The structure of the `case_when` is that the thing you're checking for goes on the left of the squiggle, and the value you want your new variable to take goes on the right. What it does is to go down the list of conditions that you are checking for, and as soon as it finds one that is true, it grabs the value on the right of the squiggle and moves on to the next row. The usual way to write these is to have a catch-all condition at the end that is always true, serving to make sure that your new variable always gets *some* value. `TRUE` is, um, always true. If you want an English word for the last condition of your `case_when`, "otherwise" is a nice one.

I wanted to check that the observer did check exactly one of `V2` and `V3` as I asserted, which can be done by gluing this onto the end:

```
bikes %>% fill(X1) %>% rename(Time = X1) %>% mutate(gender = case_when(X2 ==
    "X" & is.na(X3) ~ "Male", is.na(X2) & X3 ==
    "X" ~ "Female", TRUE ~ "Error!")) %>% count(gender)
```

```
## # A tibble: 2 x 2
##   gender     n
##   <chr>  <int>
## 1 Female   861
## 2 Male    1097
```

There are only Males and Females, so the observer really did mark exactly one X. (As a bonus, you see that there were slightly more male cyclists than female ones.)

Extra: I was wondering how `gather` would play out here. The way to do it seems to be to rename the columns first:

```
bikes %>% fill(X1) %>% rename(Time = X1) %>% rename(male = X2,
    female = X3)
```

```
## # A tibble: 1,958 x 9
##    Time male  female X4    X5    X6    X7
##    <ti> <chr> <chr>  <chr> <chr> <chr> <chr>
##  1 07:~ X     <NA>   <NA>  X     <NA>  X
##  2 07:~ X     <NA>   <NA>  X     <NA>  X
##  3 07:~ X     <NA>   <NA>  X     <NA>  X
##  4 07:~ X     <NA>   <NA>  X     <NA>  X
##  5 07:~ X     <NA>   <NA>  X     <NA>  X
##  6 07:~ X     <NA>   X     <NA>  <NA>  X
##  7 07:~ <NA>  X      X     <NA>  <NA>  X
##  8 07:~ X     <NA>   X     <NA>  <NA>  X
##  9 07:~ <NA>  X      X     <NA>  <NA>  X
## 10 07:~ X     <NA>   X     <NA>  <NA>  X
## # ... with 1,948 more rows, and 2 more
## #   variables: X8 <chr>, X9 <chr>
```

and then gather them up:

```
bikes %>% fill(X1) %>% rename(Time = X1) %>% rename(male = X2,
    female = X3) %>% gather(gender, what, male:female)
```

```
## # A tibble: 3,916 x 9
##    Time   X4    X5    X6    X7    X8    X9
##    <time> <chr> <chr> <chr> <chr> <chr> <chr>
##  1 07:00  <NA>  X     <NA>  X     <NA>  X
##  2 07:00  <NA>  X     <NA>  X     <NA>  X
##  3 07:00  <NA>  X     <NA>  X     <NA>  X
##  4 07:00  <NA>  X     <NA>  X     <NA>  X
##  5 07:00  <NA>  X     <NA>  X     <NA>  X
##  6 07:15  X     <NA>  <NA>  X     <NA>  X
##  7 07:15  X     <NA>  <NA>  X     <NA>  X
```

```
##  8 07:15  X     <NA>  <NA>  X     <NA>  X
##  9 07:15  X     <NA>  <NA>  X     X     <NA>
## 10 07:15  X     <NA>  <NA>  X     <NA>  X
## # ... with 3,906 more rows, and 2 more
## #   variables: gender <chr>, what <chr>
```

I wasn't quite sure what to call "what makes them the same", at least not until I had seen how it came out. This is where the `X` or missing goes, so we want the lines where `what` is equal to `X`:

```
bikes %>% fill(X1) %>% rename(Time = X1) %>% rename(male = X2,
    female = X3) %>% gather(gender, what, male:female) %>%
    filter(what == "X")
```

```
## # A tibble: 1,958 x 9
##    Time   X4    X5    X6    X7    X8    X9
##    <time> <chr> <chr> <chr> <chr> <chr> <chr>
##  1 07:00  <NA>  X     <NA>  X     <NA>  X
##  2 07:00  <NA>  X     <NA>  X     <NA>  X
##  3 07:00  <NA>  X     <NA>  X     <NA>  X
##  4 07:00  <NA>  X     <NA>  X     <NA>  X
##  5 07:00  <NA>  X     <NA>  X     <NA>  X
##  6 07:15  X     <NA>  <NA>  X     <NA>  X
##  7 07:15  X     <NA>  <NA>  X     <NA>  X
##  8 07:15  X     <NA>  <NA>  X     <NA>  X
##  9 07:15  X     <NA>  <NA>  X     <NA>  X
## 10 07:15  X     <NA>  <NA>  X     <NA>  X
## # ... with 1,948 more rows, and 2 more
## #   variables: gender <chr>, what <chr>
```

Another way to do this is to remove the missings in the `gather`:

```
bikes %>% fill(X1) %>% rename(Time = X1) %>% rename(male = X2,
    female = X3) %>% gather(gender, what, male:female,
    na.rm = T)
```

```
## # A tibble: 1,958 x 9
##    Time   X4    X5    X6    X7    X8    X9
##  * <time> <chr> <chr> <chr> <chr> <chr> <chr>
##  1 07:00  <NA>  X     <NA>  X     <NA>  X
##  2 07:00  <NA>  X     <NA>  X     <NA>  X
##  3 07:00  <NA>  X     <NA>  X     <NA>  X
##  4 07:00  <NA>  X     <NA>  X     <NA>  X
##  5 07:00  <NA>  X     <NA>  X     <NA>  X
##  6 07:15  X     <NA>  <NA>  X     <NA>  X
##  7 07:15  X     <NA>  <NA>  X     <NA>  X
```

```
##  8 07:15  X     <NA>  <NA>  X     <NA>  X
##  9 07:15  X     <NA>  <NA>  X     <NA>  X
## 10 07:15  X     <NA>  <NA>  X     <NA>  X
## # ... with 1,948 more rows, and 2 more
## #   variables: gender <chr>, what <chr>
```

In case you are wondering where the females went, all the males are listed first this way, and then all the females. To verify this, we can count the males and females obtained this way, and we should get the same thing we got before:

```
bikes %>% fill(X1) %>% rename(Time = X1) %>% rename(male = X2,
    female = X3) %>% gather(gender, what, male:female,
    na.rm = T) %>% count(gender)
```

```
## # A tibble: 2 x 2
##   gender     n
##   <chr>  <int>
## 1 female   861
## 2 male    1097
```

And this is what we had before.

(f) Create variables `helmet`, `passenger` and `sidewalk` in your data frame that are `TRUE` if the "Yes" column contains `X` and `FALSE` otherwise. This will use `mutate` again, but you don't need `ifelse`: just set the variable equal to the appropriate logical condition. As before, the best way to create these variables is to test the appropriate things for missingness. Note that you can create as many new variables as you like in one `mutate`. Show the first few lines of your new data frame. (Add your code onto the end of the pipe you made above.)

Solution

On the face of it, the way to do this is to go looking for `X`'s:

```
bikes %>% fill(X1) %>% rename(Time = X1) %>% mutate(gender = ifelse(is.na(X2),
    "female", "male")) %>% mutate(helmet = (X4 ==
    "X"), passenger = (X6 == "X"), sidewalk = (X8 ==
    "X"))
```

```
## # A tibble: 1,958 x 13
##    Time   X2    X3    X4    X5    X6    X7
##    <time> <chr> <chr> <chr> <chr> <chr> <chr>
##  1 07:00  X     <NA>  <NA>  X     <NA>  X
##  2 07:00  X     <NA>  <NA>  X     <NA>  X
##  3 07:00  X     <NA>  <NA>  X     <NA>  X
```

```
##  4 07:00  X     <NA>  <NA>  X     <NA>  X
##  5 07:00  X     <NA>  <NA>  X     <NA>  X
##  6 07:15  X     <NA>  X     <NA>  <NA>  X
##  7 07:15  <NA>  X     X     <NA>  <NA>  X
##  8 07:15  X     <NA>  X     <NA>  <NA>  X
##  9 07:15  <NA>  X     X     <NA>  <NA>  X
## 10 07:15  X     <NA>  X     <NA>  <NA>  X
## # ... with 1,948 more rows, and 6 more
## #   variables: X8 <chr>, X9 <chr>,
## #   gender <chr>, helmet <lgl>,
## #   passenger <lgl>, sidewalk <lgl>
```

But, we run into the same problem that we did with `gender`: the new variables are either `TRUE` or missing, never `FALSE`.

The solution is the same: look for the things that are *missing* if the cyclist is wearing a helmet, carrying a passenger or riding on the sidewalk. These are `X5, X7, X9` respectively:

```
bikes %>% fill(X1) %>% rename(Time = X1) %>% mutate(gender = ifelse(is.na(X2),
    "female", "male")) %>% mutate(helmet = is.na(X5),
    passenger = is.na(X7), sidewalk = is.na(X9))
```

```
## # A tibble: 1,958 x 13
##    Time   X2    X3    X4    X5    X6    X7
##    <time> <chr> <chr> <chr> <chr> <chr> <chr>
##  1 07:00  X     <NA>  <NA>  X     <NA>  X
##  2 07:00  X     <NA>  <NA>  X     <NA>  X
##  3 07:00  X     <NA>  <NA>  X     <NA>  X
##  4 07:00  X     <NA>  <NA>  X     <NA>  X
##  5 07:00  X     <NA>  <NA>  X     <NA>  X
##  6 07:15  X     <NA>  X     <NA>  <NA>  X
##  7 07:15  <NA>  X     X     <NA>  <NA>  X
##  8 07:15  X     <NA>  X     <NA>  <NA>  X
##  9 07:15  <NA>  X     X     <NA>  <NA>  X
## 10 07:15  X     <NA>  X     <NA>  <NA>  X
## # ... with 1,948 more rows, and 6 more
## #   variables: X8 <chr>, X9 <chr>,
## #   gender <chr>, helmet <lgl>,
## #   passenger <lgl>, sidewalk <lgl>
```

Again, you can do the `mutate` all on one line if you want to, or all four variable assignments in one `mutate`, but I used newlines and indentation to make the structure clear.

It is less elegant, though equally good for the purposes of the assignment, to use `ifelse` for these as well, which would go like this, for example:

```
bikes %>% fill(X1) %>% rename(Time = X1) %>% mutate(gender = ifelse(X2 ==
    "X", "male", "female")) %>% mutate(helmet = ifelse(is.na(X5),
    T, F))
```

```
## # A tibble: 1,958 x 11
##    Time   X2    X3    X4    X5    X6    X7
##    <time> <chr> <chr> <chr> <chr> <chr> <chr>
##  1 07:00  X     <NA>  <NA>  X     <NA>  X
##  2 07:00  X     <NA>  <NA>  X     <NA>  X
##  3 07:00  X     <NA>  <NA>  X     <NA>  X
##  4 07:00  X     <NA>  <NA>  X     <NA>  X
##  5 07:00  X     <NA>  <NA>  X     <NA>  X
##  6 07:15  X     <NA>  X     <NA>  <NA>  X
##  7 07:15  <NA>  X     X     <NA>  <NA>  X
##  8 07:15  X     <NA>  X     <NA>  <NA>  X
##  9 07:15  <NA>  X     X     <NA>  <NA>  X
## 10 07:15  X     <NA>  X     <NA>  <NA>  X
## # ... with 1,948 more rows, and 4 more
## #   variables: X8 <chr>, X9 <chr>,
## #   gender <chr>, helmet <lgl>
```

and the same for `passenger` and `sidewalk`. The warning is, whenever you see a `T` and an `F` in an `ifelse`, that you could probably get rid of the `ifelse` and use the logical condition directly. For texttt{gender}, though, you need the `ifelse` (or a `case_when`) because the values you want it to take are `male` and `female`, something other than `TRUE` and `FALSE`.

If I was helping you, and you were struggling with *ifelse* but finally mastered it, it seemed easier to suggest that you used it again for the others.

I like to put brackets around logical conditions when I am assigning them to a variable. If I don't, I get something like

```
helmet = V4 == "X"
```

which actually works, but is very hard to read. Well, I *think* it works. Let's check:

```
exes = c("X", "", "X", "", "X")
y = exes == "X"
y
```

```
## [1]  TRUE FALSE  TRUE FALSE  TRUE
```

Yes it does. But I would never recommend writing it this way, because unless you are paying attention, you won't notice which = is saving in a variable, and which one is "logically equal".

It works because of a thing called "operator precedence": the logical-equals is evaluated first, and the result of that is saved in the variable. But unless you or your readers remember that, it's better to write

```
y = (exes == "X")
```

to draw attention to the order of calculation. This is the same reason that

```
4 + 5 * 6
```

```
## [1] 34
```

evaluates this way rather than doing the addition first and getting 54. BODMAS and all that.

The `gather` approach works for these too. Rename the columns as `yes` and `no`, and then give the "what makes them the same" column a name like `helmet`. Give the "what makes them different" column a name like `what2`, to make it easier to remove later. And then do the same with the others, one `gather` at a time.

(g) Finally (for the data manipulation), get rid of all the original columns, keeping only the new ones that you created. Save the results in a data frame and display its first few rows.

Solution

This is a breath of fresh air after all the thinking needed above: this is just `select`, added to the end:

```
mybikes = bikes %>% fill(X1) %>% rename(Time = X1) %>%
    mutate(gender = ifelse(is.na(X2), "female",
        "male")) %>% mutate(helmet = is.na(X5),
    passenger = is.na(X7), sidewalk = is.na(X9)) %>%
    select(-(X2:X9))
mybikes
```

```
## # A tibble: 1,958 x 5
##    Time   gender helmet passenger sidewalk
##    <time> <chr>  <lgl>  <lgl>     <lgl>
##  1 07:00  male   FALSE  FALSE     FALSE
##  2 07:00  male   FALSE  FALSE     FALSE
##  3 07:00  male   FALSE  FALSE     FALSE
##  4 07:00  male   FALSE  FALSE     FALSE
##  5 07:00  male   FALSE  FALSE     FALSE
##  6 07:15  male   TRUE   FALSE     FALSE
##  7 07:15  female TRUE   FALSE     FALSE
##  8 07:15  male   TRUE   FALSE     FALSE
##  9 07:15  female TRUE   FALSE     TRUE
## 10 07:15  male   TRUE   FALSE     FALSE
## # ... with 1,948 more rows
```

You might not have renamed your X1, in which case, you still have it, but need to keep it (because it holds the times).

Another way to do this is to use a "select-helper", thus:

```
bikes %>% fill(X1) %>% rename(Time = X1) %>% mutate(gender = ifelse(is.na(X2),
    "female", "male")) %>% mutate(helmet = is.na(X5),
    passenger = is.na(X7), sidewalk = is.na(X9)) %>%
    select(-num_range("X", 2:9))
```

```
## # A tibble: 1,958 x 5
##    Time   gender helmet passenger sidewalk
##    <time> <chr>  <lgl>  <lgl>     <lgl>
##  1 07:00  male   FALSE  FALSE     FALSE
##  2 07:00  male   FALSE  FALSE     FALSE
##  3 07:00  male   FALSE  FALSE     FALSE
##  4 07:00  male   FALSE  FALSE     FALSE
##  5 07:00  male   FALSE  FALSE     FALSE
##  6 07:15  male   TRUE   FALSE     FALSE
##  7 07:15  female TRUE   FALSE     FALSE
##  8 07:15  male   TRUE   FALSE     FALSE
##  9 07:15  female TRUE   FALSE     TRUE
## 10 07:15  male   TRUE   FALSE     FALSE
## # ... with 1,948 more rows
```

This means "get rid of all the columns whose names are X followed by a number 2 through 9".

The pipe looks long and forbidding, but you built it (and tested it) a little at a time. Which is how you do it.

(h) The next few parts are a quick-fire analysis of the data set. They can all be solved using `count`. How many male and how many female cyclists were observed in total?

Solution

I already got this one when I was checking for observer-notation errors earlier:

```
mybikes %>% count(gender)
```

```
## # A tibble: 2 x 2
##   gender     n
##   <chr>  <int>
## 1 female   861
## 2 male    1097
```

861 females and 1097 males.

(i) How many male and female cyclists were not wearing helmets?

Solution

You can count two variables at once, in which case you get counts of all combinations of them:

```
mybikes %>% count(gender, helmet)
```

```
## # A tibble: 4 x 3
##   gender helmet     n
##   <chr>  <lgl>  <int>
## 1 female FALSE    403
## 2 female TRUE     458
## 3 male   FALSE    604
## 4 male   TRUE     493
```

403 females and 604 males were not wearing helmets, picking out what we need.

The real question of interest here is "what *proportion* of male and female cyclists were not wearing helmets?" This has a rather elegant solution that I will have to explain. First, let's go back to the `group_by` and `summarize` version of the `count` here:

But I didn't want to complicate this question any farther.

```
mybikes %>% group_by(gender, helmet) %>% summarize(the_ount = n())
```

```
## # A tibble: 4 x 3
## # Groups:   gender [?]
##   gender helmet the_ount
##   <chr>  <lgl>     <int>
## 1 female FALSE       403
## 2 female TRUE        458
## 3 male   FALSE       604
## 4 male   TRUE        493
```

That's the same table we got just now. Now, let's calculate a proportion and see what happens:

```
mybikes %>% group_by(gender, helmet) %>% summarize(the_count = n()) %>%
    mutate(prop = the_count/sum(the_count))
```

```
## # A tibble: 4 x 4
## # Groups:   gender [2]
##   gender helmet the_count  prop
##   <chr>  <lgl>      <int> <dbl>
## 1 female FALSE        403 0.468
## 2 female TRUE         458 0.532
## 3 male   FALSE        604 0.551
## 4 male   TRUE         493 0.449
```

We seem to have the proportions of males and females who were and were not wearing a helmet, and you can check that this is indeed the case, for example:

```
403/(403 + 458)
```

```
## [1] 0.4680604
```

47% of females were not wearing helmets, while 55% of males were helmetless. (You can tell from the original frequencies that a small majority of females wore helmets and a small majority of males did not.)

Now, we have to ask ourselves: how on earth did that work?

When you calculate a summary (like our `sum(count)` above), it figures that you can't want the sum by gender-helmet combination, since you already have those in `count`. You must want the sum *over* something. What? What happens is that it goes back to the `group_by` and "peels off" the last thing there, which in this case is `helmet`, leaving only `gender`. It then sums the counts for each gender, giving us what we wanted.

It just blows my mind that someone (ie., Hadley Wickham) could (i) think that this would be a nice syntax to have (instead of just being an error), (ii) find a way to implement it and (iii) find a nice logical explanation ("peeling off") to explain how it worked.

What happens if we switch the order of the things in the `group_by`?

```
mybikes %>% group_by(helmet, gender) %>% summarize(the_count = n()) %>%
    mutate(prop = the_count/sum(the_count))
```

```
## # A tibble: 4 x 4
## # Groups:   helmet [2]
##   helmet gender the_count  prop
##   <lgl>  <chr>      <int> <dbl>
## 1 FALSE  female       403 0.400
## 2 FALSE  male         604 0.600
## 3 TRUE   female       458 0.482
## 4 TRUE   male         493 0.518
```

Now we get the proportion of helmeted riders of each gender, which is not the same as what we had before. Before, we had "out of males" and "out of females"; now we have "out of helmeted riders" and "out of helmetless riders". (The riders with helmets are almost 50–50 males and females, but the riders without helmets are about 60% male.)

This is row and column proportions in a contingency table, B22 style.

Now, I have to see whether the `count` variant of this works:

```
mybikes %>% count(gender, helmet) %>% mutate(prop = n/sum(n))
```

```
## # A tibble: 4 x 4
##   gender helmet     n  prop
##   <chr>  <lgl>  <int> <dbl>
## 1 female FALSE    403 0.206
## 2 female TRUE     458 0.234
## 3 male   FALSE    604 0.308
## 4 male   TRUE     493 0.252
```

It doesn't. Well, it kind of does, but it divided by the sum of all of them rather than "peeling off", so these are overall proportions rather than row or column proportions.

So I think you have to do this the `group_by` and `summarize` way.

(j) How many cyclists were riding on the sidewalk *and* carrying a passenger?

Solution

Not too many, I'd hope. Again:

```
mybikes %>% count(passenger, sidewalk)
```

```
## # A tibble: 3 x 3
##   passenger sidewalk     n
##   <lgl>     <lgl>    <int>
## 1 FALSE     FALSE     1880
## 2 FALSE     TRUE        73
## 3 TRUE      FALSE        5
```

We're looking for the "true", "true" entry of that table, which seems to have vanished. That means the count is *zero*: none at all. (There were only 5 passenger-carrying riders, and they were all on the road.)

(k) What was the busiest 15-minute period of the day, and how many cyclists were there then?

Solution

The obvious way is to list every 15-minute period and eyeball the largest frequency. There are quite a few 15-minute periods, so be prepared to hit Next a few times (or use `View`):

```
mybikes %>% count(Time) %>% print(n = Inf)
```

```
## # A tibble: 48 x 2
##    Time       n
##    <time> <int>
```

```
##  1 07:00      5
##  2 07:15      8
##  3 07:30      9
##  4 07:45      5
##  5 08:00     18
##  6 08:15     14
##  7 08:30     12
##  8 08:45     22
##  9 09:00     17
## 10 09:15     15
## 11 09:30     16
## 12 09:45     12
## 13 10:00      8
## 14 10:15     21
## 15 10:30     22
## 16 10:45     19
## 17 11:00     30
## 18 11:15     14
## 19 11:30     27
## 20 11:45     28
## 21 12:00     22
## 22 12:15     35
## 23 12:30     40
## 24 12:45     33
## 25 13:00     44
## 26 13:15     46
## 27 13:30     29
## 28 13:45     41
## 29 14:00     36
## 30 14:15     45
## 31 14:30     33
## 32 14:45     50
## 33 15:00     49
## 34 15:15     51
## 35 15:30     61
## 36 15:45     79
## 37 16:00    101
## 38 16:15     56
## 39 16:30     51
## 40 16:45     75
## 41 17:00    104
## 42 17:15    128
## 43 17:30     80
## 44 17:45     73
```

```
## 45 18:00     75
## 46 18:15     78
## 47 18:30     63
## 48 18:45     58
```

17:15, or 5:15 pm, with 128 cyclists.

But, computers are meant to save us that kind of effort. How? Note that the output from `count` is itself a data frame, so anything you can do to a data frame, you can do to *it*: for example, display only the rows where the frequency equals the maximum frequency:

```
mybikes %>% count(Time) %>% filter(n == max(n))
```

```
## # A tibble: 1 x 2
##   Time       n
##   <time> <int>
## 1 17:15    128
```

That will actually display *all* the times where the cyclist count equals the maximum, of which there might be more than one.

12.11 *Feeling the heat*

In summer, the city of Toronto issues Heat Alerts for "high heat or humidity that is expected to last two or more days". The precise definitions are shown at link. During a heat alert, the city opens Cooling Centres and may extend the hours of operation of city swimming pools, among other things. All the heat alert days from 2001 to 2016 are listed at link.

The word "warning" is sometimes used in place of "alert" in these data. They mean the same thing.

Unlike *thunderstorm watch* and *thunderstorm warning*, which mean different things.

(a) Read the data into R, and display the data frame. Note that there are four columns:

- a numerical `id` (numbered upwards from the first Heat Alert in 2001; some of the numbers are missing)
- the `date` of the heat alert, in year-month-day format with 4-digit years.
- a text `code` for the type of heat alert
- `text` describing the kind of heat alert. This can be quite long.

Solution

A `.csv`, so:

```
my_url = "http://www.utsc.utoronto.ca/~butler/c32/heat.csv"
heat = read_csv(my_url)
```

```
## Parsed with column specification:
## cols(
##   id = col_integer(),
##   date = col_date(format = ""),
##   code = col_character(),
##   text = col_character()
## )
```

```
heat
```

```
## # A tibble: 200 x 4
##       id date       code  text
##    <int> <date>     <chr> <chr>
##  1   232 2016-09-08 HAU   Toronto's Medical~
##  2   231 2016-09-07 HAE   Toronto's Medical~
##  3   230 2016-09-06 HA    Toronto's Medical~
##  4   228 2016-08-13 EHAE  Toronto's Medical~
##  5   227 2016-08-12 EHAE  Toronto's Medical~
##  6   226 2016-08-11 HAU   Toronto's Medical~
##  7   225 2016-08-10 HAE   Toronto's Medical~
##  8   224 2016-08-09 HA    Toronto's Medical~
##  9   222 2016-08-05 HAE   Toronto's Medical~
## 10   221 2016-08-04 HA    Toronto's Medical~
## # ... with 190 more rows
```

You might get a truncated `text` as I did, or you might have to click to see more of it. In any case, we won't be using the text, so you can just forget about it from here on.

- In your data frame, are the dates stored as genuine dates or as text? How can you tell?

Solution

Look at the top of the column on your display of the data frame. Under the date column it says `date` rather than `chr` (which means "text"), so these are genuine dates. This happened because the data file contained the dates in year-month-day order, so `read_csv` read them in as dates. (If they had been in some other order, they would have been read in as text and we would need to use `lubridate` to make them into dates.)

- Which different heat alert codes do you have, and how many of each?

Solution

`count`, most easily:

```
heat %>% count(code)
```

```
## # A tibble: 6 x 2
##   code      n
##   <chr> <int>
## 1 EHA      59
## 2 EHAD      1
## 3 EHAE     18
## 4 HA       93
## 5 HAE      16
## 6 HAU      13
```

Alternatively, `group_by` and `summarize`:

```
heat %>% group_by(code) %>% summarize(count = n())
```

```
## # A tibble: 6 x 2
##   code  count
##   <chr> <int>
## 1 EHA      59
## 2 EHAD      1
## 3 EHAE     18
## 4 HA       93
## 5 HAE      16
## 6 HAU      13
```

(note that `n()` gives the number of rows, in each group if you have groups.)

There are six different codes, but EHAD only appears once.

(b) Use the `text` in your dataset (or look back at the original data file) to describe briefly in your own words what the various codes represent.

Solution

You can check that each time a certain code appears, the text next to it is identical. The six codes and my brief descriptions are:

EHA (Start of) Extended Heat Alert

EHAD Extreme Heat Alert downgraded to Heat Alert

EHAE Extended Heat Alert continues

HA (Start of) Heat Alert

HAE Heat Alert continues

HAU Heat Alert upgraded to Extended Heat Alert

I thought there was such a thing as an Extreme Heat Alert, but here the word is (usually) Extended, meaning a heat alert that extends over several days, long in duration rather than extremely hot. The only place Extreme occurs is in EHAD, which only occurs once. I want your answer to say or suggest something about whether a code applies *only* to continuing heat alerts (ie., that EHAD, EHAE, HAE and HAU are different from the others).

(c) How many (regular and extended) heat alert events are there altogether? A heat alert event is a stretch of consecutive days, on all of which there is a heat alert or extended heat alert. Hints: (i) you can answer this from output you already have; (ii) how can you tell when a heat alert event *starts*?

Solution

This turned out to be more messed-up than I thought. There is a detailed discussion below. The codes EHAD, EHAE, HAE, HAU all indicate that there was a heat alert on the day before. Only the codes HA and EHA can indicate the start of a heat alert (event). The problem is that HA and EHA sometimes indicate the start of a heat alert event and sometimes one that is continuing. You can check by looking at the data that HA and EHA days can (though they don't always: see below) have a non-heat-alert day before (below) them in the data file: for example, August 4, 2012 is an HA day, but August 3 of that year was not part of any kind of heat alert. I had intended the answer to be this:

> So we get the total number of heat alert events by totalling up the number of HA and EHA days: $59 + 93 = 152$.

This is not right because there are some consecutive EHA days, eg. 5–8 July 2010, so that EHA sometimes indicates the continuation of an extended heat alert and sometimes the start of one. I was expecting EHA to be used only for the start, and one of the other codes to indicate a continuation. The same is (sometimes) true of HA. So reasonable answers to the question as set include:

- 93, the number of HAs
- 59, the number of EHAs
- 152, the number of HAs and EHAs combined
- "152 or less", "between 93 and 152", "between 59 and 152" to reflect that not all of these mark the start of a heat alert event.

Any of these, or something similar *with an explanation of how you got your answer*, are acceptable. In your career as a data scientist, you will often run into this kind of thing, and it will be your job to do something with the data and *explain* what you did so that somebody else can decide whether they believe you or not. A good explanation, even if it is not correct, will help you get at the truth because it will inspire someone to say ‘`in fact, it goes *this* way'', and then the two of you can jointly figure out what's actually going on. Detailed discussion follows. If you have *any* ambitions of working with data, you should try to follow the paragraphs below, because they indicate how you would get an *actual* answer to the question. I used R and`dplyr`, because that seemed to be the easiest route. I think the key is the number of days between one heat alert day and the next one.`dplyr`has a `function`diff’ that works out exactly this. Building a pipeline, just because:

```
heat %>% select(-text) %>% mutate(daycount = as.numeric(date)) %>%
    mutate(daydiff = abs(c(diff(daycount), 0)))
```

```
## # A tibble: 200 x 5
##       id date       code  daycount daydiff
##    <int> <date>     <chr>    <dbl>   <dbl>
##  1   232 2016-09-08 HAU      17052       1
##  2   231 2016-09-07 HAE      17051       1
##  3   230 2016-09-06 HA       17050      24
##  4   228 2016-08-13 EHAE     17026       1
##  5   227 2016-08-12 EHAE     17025       1
##  6   226 2016-08-11 HAU      17024       1
##  7   225 2016-08-10 HAE      17023       1
##  8   224 2016-08-09 HA       17022       4
##  9   222 2016-08-05 HAE      17018       1
## 10   221 2016-08-04 HA       17017      11
## # ... with 190 more rows
```

Oof. I have some things to keep track of here:

- Get rid of the `text`, since it serves no purpose here.
- The `date` column is a proper `Date` (I checked).
- Then I want the date as number of days; since it is a number of days internally, I just make it a number with `as.numeric`.
- Then I use `diff` to get the difference between each date and the previous one, remembering to glue a 0 onto the end so that I have the right number of differences.

- Since the dates are most recent first, I take the absolute value so that the `daydiff` values are positive (except for the one that is 0 on the end).

Still with me? All right. You can check that the `daydiff` values are the number of days between the date on that line and the line below it. For example, there were 24 days between August 13 and September 6.

Now, when `daydiff` is 1, there was also a heat alert on the previous day (the line below in the file), but when `daydiff` is *not* 1, that day must have been the *start* of a heat alert event. So if I count the non-1's, that will count the number of heat alert events there were. (That includes the difference of 0 on the first day, the one at the end of the file.)

Thus my pipeline continues like this:

```
heat %>% select(-text) %>% mutate(daycount = as.numeric(date)) %>%
    mutate(daydiff = abs(c(diff(daycount), 0))) %>%
    count(daydiff != 1)
```

```
## # A tibble: 2 x 2
##   `daydiff != 1`     n
##   <lgl>          <int>
## 1 FALSE            121
## 2 TRUE              79
```

And that's how many actual heat alert events there were: 79, less even than the number of HAs. So that tells me that a lot of my HAs and EHAs were actually continuations of heat alert events rather than the start of them. I think I need to have a word with the City of Toronto about their data collection processes.

`count` will count anything that is, or can be made into, a categorical variable. It doesn't have to be one of the columns of your data frame; here it is something that is either `TRUE` or `FALSE` about every row of the data frame.

One step further: what *is* the connection between the codes and the start of heat alert events? We can figure that out now:

```
heat %>% select(-text) %>% mutate(daycount = as.numeric(date)) %>%
    mutate(daydiff = abs(c(diff(daycount), 0))) %>%
    mutate(start = (daydiff != 1)) %>% count(code,
    start)
```

```
## # A tibble: 8 x 3
##   code  start     n
##   <chr> <lgl> <int>
```

```
## 1 EHA   FALSE    53
## 2 EHA   TRUE      6
## 3 EHAD  FALSE     1
## 4 EHAE  FALSE    18
## 5 HA    FALSE    20
## 6 HA    TRUE     73
## 7 HAE   FALSE    16
## 8 HAU   FALSE    13
```

I made a column `start` that is `TRUE` at the start of a heat alert event and `FALSE` otherwise, by comparing the days from the previous heat alert day with 1. Then I can make a `table`, or, as here, the `dplyr` equivalent with `count`. Or `group_by` and `summarize`. What this shows is that EHAD, EHAE, HAE and HAU *never* go with the start of a heat alert event (as they shouldn't). But look at the HAs and EHAs. For the HAs, 73 of them go with the start of an event, but 20 do not. For the EHAs, just 6 of them go with the start, and 53 do not. (Thus, counting just the HAs was very much a reasonable thing to do.)

I did not know until just now that you could put two variables in a count and you get counts of all the combinations of them. Just goes to show the value of *try it and see*.

The 79 heat alert events that we found above had 73 of them starting with an HA, and just 6 starting with an EHA. I wasn't quite sure how this would come out, but I knew it had something to do with the number of days between one heat alert day and the next, so I calculated those first and then figured out what to do with them. here down is SAS

(d) We are going to investigate how many heat alert days there were in each year. To do that, we have to extract the year from each of our dates.

Solution
This will need `lubridate`:

```
library(lubridate)
```

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

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

```
heat %>% select(-text) %>% mutate(year = year(date)) %>%
    sample_n(10)
```

```
## # A tibble: 10 x 4
##       id date       code   year
```

```
##    <int> <date>     <chr> <dbl>
##  1    48 2005-07-11 EHA    2005
##  2   131 2011-07-22 EHAE   2011
##  3   197 2015-08-16 HA     2015
##  4   114 2010-07-08 EHA    2010
##  5    47 2005-07-10 HA     2005
##  6    62 2006-05-31 EHA    2006
##  7   103 2009-08-17 HA     2009
##  8   143 2012-06-21 EHAE   2012
##  9    35 2005-06-07 HA     2005
## 10    93 2008-06-09 EHA    2008
```

That seems to have worked. I listed a random sample of rows to get back to previous years. Having convinced myself that it worked, let me save it:

```
heat <- heat %>% select(-text) %>% mutate(year = year(date))
```

(e) Count the number of heat alert days for each year, by tabulating the year variable. Looking at this table, would you say that there have been more heat alert days in recent years? Explain (very) briefly.

Solution
Count them again:

```
heat %>% count(year)
```

```
## # A tibble: 16 x 2
##     year     n
##    <dbl> <int>
##  1  2001     9
##  2  2002    16
##  3  2003     6
##  4  2004     2
##  5  2005    26
##  6  2006    17
##  7  2007    15
##  8  2008     9
##  9  2009     3
## 10  2010    16
## 11  2011    12
## 12  2012    21
## 13  2013    13
## 14  2014     1
## 15  2015    12
## 16  2016    22
```

There are various things you could say, most of which are likely to be good. My immediate reaction is that most of the years with a lot of heat-alert days are in the last few years, and most of the years with not many are near the start, so there is something of an upward trend. Having said that, 2014 is unusually low (that was a cool summer, if you recall), and 2005 was unusually high. (Was that the summer of the big power outage? I forget.)

I looked it up. It was 2003, my first summer in Ontario.

You could also reasonably say that there isn't much pattern: the number of heat-alert days goes up and down. In fact, anything that's not obviously nonsense will do.

I was thinking about making a graph of these frequencies against year, and sticking some kind of smooth trend on it. This uses the output we just got, which is itself a data frame:

```
heat %>% count(year) %>% ggplot(aes(x = year,
    y = n)) + geom_point() + geom_smooth(se = F)
```

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

The pattern is very scattered, as is commonly the case with environmental-science-type data, but there is a very small upward trend. So it seems that either answer is justified, either "there is no trend" or "there is something of an upward trend".

The other thing I notice on this plot is that if there are a lot of heat-alert days one year, there will probably also be a lot in the next year (and correspondingly if the number of heat-alert days is below average: it tends to be below average again in the next year). This pattern is known to time-series people as "autocorrelation" and indicates that the number of heat-alert days in one year and the next is not independent: if you know one year, you can predict the next year. (Assessment of trend and autocorrelation are hard to untangle properly.)

I learn from Environmental Science grad students (of whom we have a number at UTSC) that the approved measure of association is called the Mann-Kendall correlation, which is the Kendall correlation of the data values with time. In the same way that we use the sign test when we doubt normality, and it uses the data more crudely but safely, the regular (so-called Pearson) correlation assumes normality (of the errors in the regression of one variable on the other), and when you doubt that (as you typically do with this kind of data) you compute a different kind of correlation with time. What the Kendall correlation does is to take each pair of observations and ask whether the trend with time is uphill or downhill. For example, there were 3 heat-alert days in 2009, 16 in 2010 and 12 in 2011. Between 2009 and 2010, the trend is uphill (increasing with time), and also between 2009 and 2011 (there were more heat-alert days in the later year), but between 2010 and 2011 the trend is downhill. The idea of the Kendall correlation is you take *all* the pairs of points, of which there are typically rather a lot, count up how many pairs are uphill and how many downhill, and apply a formula to get a correlation between -1 and 1. (If there are about an equal number of uphills and downhills, the correlation comes out near 0; if they are mostly uphill, the correlation is near 1, and if they are mostly downhill, the correlation is near -1.) It doesn't matter *how* uphill or downhill the trends are, only the number of each, in the same way that the sign test only counts the *number* of values above or below the hypothesized median, not how far above or below they are.

This can be calculated, and even tested:

```
heat %>% count(year) %>% with(., cor.test(year,
    n, method = "kendall"))
```

```
## Warning in cor.test.default(year, n, method =
## "kendall"): Cannot compute exact p-value with
## ties
```

```
##
##  Kendall's rank correlation tau
##
## data:  year and n
## z = 0.31612, p-value = 0.7519
## alternative hypothesis: true tau is not equal to 0
## sample estimates:
##        tau
## 0.05907646
```

The Mann-Kendall correlation is a thoroughly unremarkable 0.06, and with only 16 data points, a null hypothesis that the correlation is

zero is far from being rejected, P-value 0.7519 as shown. So this is no evidence of a time trend at all.

I'd like to say a word about how I got these data. They came from link. If you take a look there, there are no obvious rows and columns. This format is called JSON. Look a bit more carefully and you'll see stuff like this, repeated:

```
{"id":"232","date":"2016-09-08","code":"HAU",
"text":"Toronto's Medical Officer of Health has upgraded the Heat Warning to an Extended Heat Warning"}
```

one for each heat alert day. These are "keys" (on the left side of the `:`) and "values" (on the right side). The keys are column headers (if the data were in a data frame) and the values are the data values that would be in that column. In JSON generally, there's no need for the keys to be the same in every row, but if they are, as they are here, the data can be arranged in a data frame. How? Read on.

This is the same kind of thing as a *dictionary* in Python.

I did this in R, using a package called `jsonlite`, with this code:

```
library(jsonlite)
url = "http://app.toronto.ca/opendata/heat_alerts/heat_alerts_list.json"
heat = fromJSON(url, simplifyDataFrame = T)
head(heat)
write_csv(heat, "heat.csv")
```

After loading the package, I create a variable `url` that contains the URL for the JSON file. The `fromJSON` line takes something that is JSON (which could be text, a file or a URL) and converts it to and saves it in a data frame. Finally, I save the data frame in a `.csv` file. That's the `.csv` file you used. If you run that code, you'll get a `.csv` file of heat alerts right up to the present, and you can update my analysis.

Why `.csv`? If I had used `write_delim`, the values would have been separated by spaces. *But*, the `text` is a sentence of several words, which are themselves separated by spaces. I could have had you read in everything else and not the text, and then separated-by-spaces would have been fine, but I wanted you to see the text so that you could understand the `code` values. So `.csv` is what it was.

13
Regression

```
library(tidyverse)
```

```
## -- Attaching packages ---- tidyverse 1.2.1 --
```

```
## v ggplot2 3.1.0     v purrr   0.2.5
## v tibble  1.4.2     v dplyr   0.7.8
## v tidyr   0.8.1     v stringr 1.3.1
## v readr   1.1.1     v forcats 0.3.0
```

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

These problems are about simple regression (just one x-variable):

13.1 *Rainfall in California*

The data in link are rainfall and other measurements for 30 weather stations in California. Our aim is to understand how well the annual rainfall at these stations (measured in inches) can be predicted from the other measurements, which are the altitude (in feet above sea level), the latitude (degrees north of the equator) and the distance from the coast (in miles).

(a) Read the data into R. You'll have to be careful here, since the values are space-delimited, but sometimes by more than one space, to make the columns line up. `read_table2`, with filename or url, will read it in. One of the variables is called `rainfall`, so as long as you *do not* call the data frame that, you should be safe.

Solution
I used `rains` as the name of my data frame:

```
my_url = "http://www.utsc.utoronto.ca/~butler/c32/calirain.txt"
rains = read_table2(my_url)
```

```
## Parsed with column specification:
## cols(
##   station = col_character(),
##   rainfall = col_double(),
##   altitude = col_integer(),
##   latitude = col_double(),
##   fromcoast = col_integer()
## )
```

I have the right number of rows and columns.

There is also `read_table`, but that requires *all* the columns, including the header row, to be lined up. You can try that here and see how it fails.

I don't need you to investigate the data yet (that happens in the next part), but this is interesting (to me):

```
rains
```

```
## # A tibble: 30 x 5
##    station rainfall altitude latitude
##    <chr>      <dbl>    <int>    <dbl>
##  1 Eureka      39.6       43     40.8
##  2 RedBlu~     23.3      341     40.2
##  3 Thermal     18.2     4152     33.8
##  4 FortBr~     37.5       74     39.4
##  5 SodaSp~     49.3     6752     39.3
##  6 SanFra~     21.8       52     37.8
##  7 Sacram~     18.1       25     38.5
##  8 SanJose     14.2       95     37.4
##  9 GiantF~     42.6     6360     36.6
## 10 Salinas     13.8       74     36.7
## # ... with 20 more rows, and 1 more
## #   variable: fromcoast <int>
```

Some of the station names are two words, but they have been smooshed into one word, so that `read_table2` will recognize them as a single thing. Someone had already done that for us, so I didn't even have to do it myself.

If the station names had been two genuine words, a `.csv` would probably have been the best choice (the actual data values being separated by commas then, and not spaces).

(b) Make a boxplot of the rainfall figures, and explain why the values are reasonable. (A rainfall cannot be negative, and it is unusual for a annual rainfall to exceed 60 inches.) A `ggplot` boxplot needs *something* on the x-axis: the number 1 will do.

Solution

```
ggplot(rains, aes(y = rainfall, x = 1)) + geom_boxplot()
```

There is only one rainfall over 60 inches, and the smallest one is close to zero but positive, so that is good.

Another possible plot here is a histogram, since there is only one quantitative variable:

```
ggplot(rains, aes(x = rainfall)) + geom_histogram(bins = 7)
```

This clearly shows the rainfall value above 60 inches, but some other things are less clear: are those two rainfall values around 50 inches above or below 50, and are those six rainfall values near zero actually above zero? Extra: What stations have those extreme values? Should you wish to find out:

```
rains %>% filter(rainfall > 60)
```

```
## # A tibble: 1 x 5
##   station rainfall altitude latitude
##   <chr>      <dbl>    <int>    <dbl>
## 1 Cresce~     74.9       35     41.7
## # ... with 1 more variable: fromcoast <int>
```

This is a place right on the Pacific coast, almost up into Oregon (it's almost the northernmost of all the stations). So it makes sense that it would have a high rainfall, if anywhere does. (If you know anything about rainy places, you'll probably think of Vancouver and Seattle, in the Pacific Northwest.) Here it is: link. Which station has less than 2 inches of annual rainfall?

```
rains %>% filter(rainfall < 2)
```

```
## # A tibble: 1 x 5
##   station rainfall altitude latitude
##   <chr>      <dbl>    <int>    <dbl>
## 1 DeathV~     1.66     -178     36.5
## # ... with 1 more variable: fromcoast <int>
```

The name of the station is a clue: this one is in the desert. So you'd expect very little rain. Its altitude is *negative*, so it's actually *below* sea level. This is correct. Here is where it is: link.

(c) Plot `rainfall` against each of the other quantitative variables (that is, not `station`).

Solution

That is, `altitude`, `latitude` and `fromcoast`. The obvious way to do this (perfectly acceptable) is one plot at a time:

```
ggplot(rains, aes(y = rainfall, x = altitude)) +
    geom_point()
```

```
ggplot(rains, aes(y = rainfall, x = latitude)) +
    geom_point()
```

and finally

```
ggplot(rains, aes(y = rainfall, x = fromcoast)) +
    geom_point()
```

You can add a smooth trend to these if you want. Up to you. Just the points is fine with me.

Here is a funky way to get all three plots in one shot:

```
rains %>% gather(xname, x, altitude:fromcoast) %>%
    ggplot(aes(x = x, y = rainfall)) + geom_point() +
    facet_wrap(~xname, scales = "free")
```

This always seems extraordinarily strange if you haven't run into it before. The strategy is to put *all* the x-variables you want to plot into *one* column and then plot your y against the x-column. A nice side-effect of the way `gather` works is that what makes the x-columns different is that they are x-variables with different *names*, which is exactly what you want later for the facets. Thus: make a column of all the x's glued together, labelled by which x they are, then plot y against x but make a different sub-plot or "facet" for each different x-name. The last thing is that each x is measured on a different scale, and unless we take steps, all the sub-plots will have the *same* scale on each axis, which we don't want.

I'm not sure I like how it came out, with three very tall plots. `facet_wrap` can also take an `nrow` or an `ncol`, which tells it how many rows or columns to use for the display. Here, for example, two columns because I thought three was too many:

```
rains %>% gather(xname, x, altitude:fromcoast) %>%
    ggplot(aes(x = x, y = rainfall)) + geom_point() +
    facet_wrap(~xname, scales = "free", ncol = 2)
```

Now, the three plots have come out about square, which I like a lot better.

(d) Look at the relationship of each other variable with `rainfall`. Justify the assertion that `latitude` seems most strongly related with `rainfall`. Is that relationship positive or negative? linear? Explain briefly.

Solution
Let's look at the three variables in turn:

- `altitude`: not much of anything. The stations near sea level have rainfall all over the place, though the three highest-altitude stations have the three highest rainfalls apart from Crescent City.
- `latitude`: there is a definite upward trend here, in that stations further north (higher latitude) are likely to have a higher rainfall. I'd call this trend linear (or, not obviously curved), though the two most northerly stations have one higher and one much lower rainfall than you'd expect.
- `fromcoast`: this is a weak downward trend, though the trend is spoiled by those three stations about 150 miles from the coast that have more than 40 inches of rainfall.

Out of those, only `latitude` seems to have any meaningful relationship with `rainfall`.

(e) Fit a regression with `rainfall` as the response variable, and `latitude` as your explanatory variable. What are the intercept, slope and R-squared values? Is there a *significant* relationship between `rainfall` and your explanatory variable? What does that mean?

Solution

Save your `lm` into a variable, since it will get used again later:

```
rainfall.1 = lm(rainfall ~ latitude, data = rains)
summary(rainfall.1)
```

```
##
## Call:
## lm(formula = rainfall ~ latitude, data = rains)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -27.297  -7.956  -2.103   6.082  38.262
##
## Coefficients:
##              Estimate Std. Error t value
## (Intercept) -113.3028    35.7210  -3.172
## latitude       3.5950     0.9623   3.736
##             Pr(>|t|)
## (Intercept)  0.00366 **
## latitude     0.00085 ***
## ---
## Signif. codes:
##   0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1  ' ' 1
##
## Residual standard error: 13.82 on 28 degrees of freedom
## Multiple R-squared:  0.3326, Adjusted R-squared:  0.3088
## F-statistic: 13.96 on 1 and 28 DF,  p-value: 0.0008495
```

My intercept is -113.3, slope is 3.6 and R-squared is 0.33 or 33%. (I want you to pull these numbers out of the output and round them off to something sensible.) The slope is significantly nonzero, its P-value being 0.00085: rainfall really does depend on latitude, although not strongly so.

Extra: Of course, I can easily do the others as well, though you don't have to:

```
rainfall.2 = lm(rainfall ~ fromcoast, data = rains)
summary(rainfall.2)
```

```
##
## Call:
## lm(formula = rainfall ~ fromcoast, data = rains)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
```

```
## -15.240  -9.431  -6.603   2.871  51.147
##
## Coefficients:
##             Estimate Std. Error t value
## (Intercept) 23.77306    4.61296   5.154
## fromcoast   -0.05039    0.04431  -1.137
##             Pr(>|t|)
## (Intercept) 1.82e-05 ***
## fromcoast      0.265
## ---
## Signif. codes:
##   0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1  ' ' 1
##
## Residual standard error: 16.54 on 28 degrees of freedom
## Multiple R-squared:  0.04414,    Adjusted R-squared:   0.01
## F-statistic: 1.293 on 1 and 28 DF,  p-value: 0.2651
```

Here, the intercept is 23.8, the slope is -0.05 and R-squared is a dismal 0.04 (4%). This is a way of seeing that this relationship is really weak, and it doesn't even have a curve to the trend or anything that would compensate for this. I looked at the scatterplot again and saw that if it were not for the point bottom right which is furthest from the coast and has almost no rainfall, there would be almost no trend at all. The slope here is not significantly different from zero, with a P-value of 0.265.

Finally:

```r
rainfall.3 = lm(rainfall ~ altitude, data = rains)
summary(rainfall.3)
```

```
##
## Call:
## lm(formula = rainfall ~ altitude, data = rains)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -20.620  -8.479  -2.729   4.555  58.271
##
## Coefficients:
##              Estimate Std. Error t value
## (Intercept) 16.514799   3.539141   4.666
## altitude     0.002394   0.001428   1.676
##             Pr(>|t|)
## (Intercept)  6.9e-05 ***
## altitude       0.105
```

```
## ---
## Signif. codes:
##   0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1  ' ' 1
##
## Residual standard error: 16.13 on 28 degrees of freedom
## Multiple R-squared:  0.09121,    Adjusted R-squared:  0.05875
## F-statistic:  2.81 on 1 and 28 DF,  p-value: 0.1048
```

The intercept is 16.5, the slope is 0.002 and the R-squared is 0.09 or 9%, also terrible. The P-value is 0.105, which is not small enough to be significant.

So it looks as if it's only `latitude` that has any impact at all. This is the only explanatory variable with a significantly nonzero slope. On its own, at least.

(f) Fit a multiple regression predicting `rainfall` from all three of the other (quantitative) variables. Display the results. Comment is coming up later.

Solution

This, then:

```r
rainfall.4 = lm(rainfall ~ latitude + altitude +
    fromcoast, data = rains)
summary(rainfall.4)
```

```
##
## Call:
## lm(formula = rainfall ~ latitude + altitude + fromcoast, data = rains)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -28.722  -5.603  -0.531   3.510  33.317
##
## Coefficients:
##               Estimate Std. Error t value
## (Intercept) -1.024e+02  2.921e+01  -3.505
## latitude     3.451e+00  7.949e-01   4.342
## altitude     4.091e-03  1.218e-03   3.358
## fromcoast   -1.429e-01  3.634e-02  -3.931
##             Pr(>|t|)
## (Intercept) 0.001676 **
## latitude    0.000191 ***
## altitude    0.002431 **
## fromcoast   0.000559 ***
## ---
```

```
## Signif. codes:
##   0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1  ' ' 1
##
## Residual standard error: 11.1 on 26 degrees of freedom
## Multiple R-squared:  0.6003, Adjusted R-squared:  0.5542
## F-statistic: 13.02 on 3 and 26 DF,  p-value: 2.205e-05
```

(g) What is the R-squared for the regression of the last part? How does that compare with the R-squared of your regression in part (e)?

Solution

The R-squared is 0.60 (60%), which is quite a bit bigger than the R-squared of 0.33 (33%) we got back in (e).

(h) What do you conclude about the importance of the variables that you did *not* include in your model in (e)? Explain briefly.

Solution

Both variables `altitude` and `fromcoast` are significant in this regression, so they have *something to add* over and above `latitude` when it comes to predicting rainfall, even though (and this seems odd) they have no apparent relationship with `rainfall` on their own. Another way to say this is that the three variables work together as a team to predict rainfall, and together they do much better than any one of them can do by themselves.

This also goes to show that the scatterplots we began with don't get to the heart of multi-variable relationships, because they are only looking at the variables two at a time.

(i) Make a suitable hypothesis test that the variables `altitude` and `fromcoast` significantly improve the prediction of `rainfall` over the use of `latitude` alone. What do you conclude?

Solution

This calls for `anova`. Feed this two fitted models, smaller (fewer explanatory variables) first. The null hypothesis is that the two models are equally good (so we should go with the smaller); the alternative is that the larger model is better, so that the extra complication is worth it:

```
anova(rainfall.1, rainfall.4)
```

```
## Analysis of Variance Table
##
## Model 1: rainfall ~ latitude
## Model 2: rainfall ~ latitude + altitude + fromcoast
```

```
##   Res.Df    RSS Df Sum of Sq      F   Pr(>F)
## 1     28 5346.8
## 2     26 3202.3  2    2144.5 8.7057 0.001276
##
## 1
## 2 **
## ---
## Signif. codes:
##   0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1  ' ' 1
```

The P-value is small, so we reject the null in favour of the alternative: the regression with all three explanatory variables fits better than the one with just `latitude`, so the bigger model is the one we should go with.

If you have studied these things: this one is a "multiple-partial F-test", for testing the combined significance of more than one x but less than all the x's.

If you had just one x, you'd use a t-test for its slope, and if you were testing all the x's, you'd use the global F-test that appears in the regression output.

13.2 *Carbon monoxide in cigarettes*

The (US) Federal Trade Commission assesses cigarettes according to their tar, nicotine and carbon monoxide contents. In a particular year, 25 brands were assessed. For each brand, the tar, nicotine and carbon monoxide (all in milligrams) were measured, along with the weight in grams. Our aim is to predict carbon monoxide from any or all of the other variables. The data are in link. These are aligned by column (except for the variable names), with more than one space between each column of data.

(a) Read the data into R, and check that you have 25 observations and 4 variables.

Solution

This specification calls for `read_table2`:

```r
my_url = "http://www.utsc.utoronto.ca/~butler/c32/ftccigar.txt"
cigs = read_table2(my_url)
```

```
## Parsed with column specification:
## cols(
##   tar = col_double(),
##   nicotine = col_double(),
##   weight = col_double(),
##   co = col_double()
## )
```

```
cigs
```

```
## # A tibble: 25 x 4
##      tar nicotine weight    co
##    <dbl>    <dbl>  <dbl> <dbl>
##  1  14.1     0.86  0.985  13.6
##  2  16       1.06  1.09   16.6
##  3  29.8     2.03  1.16   23.5
##  4   8       0.67  0.928  10.2
##  5   4.1     0.4   0.946   5.4
##  6  15       1.04  0.888  15
##  7   8.8     0.76  1.03    9
##  8  12.4     0.95  0.922  12.3
##  9  16.6     1.12  0.937  16.3
## 10  14.9     1.02  0.886  15.4
## # ... with 15 more rows
```

Yes, I have 25 observations on 4 variables indeed.

`read_delim` won't work (try it and see what happens), because that would require the values to be separated by *exactly one* space.

(b) Run a regression to predict carbon monoxide from the other variables, and obtain a summary of the output.

Solution

The word "summary" is meant to be a big clue that `summary` is what you need:

```
cigs.1 = lm(co ~ tar + nicotine + weight, data = cigs)
summary(cigs.1)
```

```
##
## Call:
## lm(formula = co ~ tar + nicotine + weight, data = cigs)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -2.89261 -0.78269  0.00428  0.92891  2.45082
##
## Coefficients:
##             Estimate Std. Error t value
## (Intercept)   3.2022     3.4618   0.925
## tar           0.9626     0.2422   3.974
## nicotine     -2.6317     3.9006  -0.675
## weight       -0.1305     3.8853  -0.034
##             Pr(>|t|)
```

```
## (Intercept) 0.365464
## tar         0.000692 ***
## nicotine    0.507234
## weight      0.973527
## ---
## Signif. codes:
##   0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1  ' ' 1
##
## Residual standard error: 1.446 on 21 degrees of freedom
## Multiple R-squared:  0.9186, Adjusted R-squared:  0.907
## F-statistic: 78.98 on 3 and 21 DF,  p-value: 1.329e-11
```

(c) Which one of your explanatory variables would you remove from this regression? Explain (very) briefly. Go ahead and fit the regression without it, and describe how the change in R-squared from the regression in (b) was entirely predictable.

Solution

First, the x-variable to remove. The obvious candidate is `weight`, since it has easily the highest, and clearly non-significant, P-value. So, out it comes:

```r
cigs.2 = lm(co ~ tar + nicotine, data = cigs)
summary(cigs.2)
```

```
##
## Call:
## lm(formula = co ~ tar + nicotine, data = cigs)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -2.89941 -0.78470 -0.00144  0.91585  2.43064
##
## Coefficients:
##             Estimate Std. Error t value
## (Intercept)   3.0896     0.8438   3.662
## tar           0.9625     0.2367   4.067
## nicotine     -2.6463     3.7872  -0.699
##             Pr(>|t|)
## (Intercept) 0.001371 **
## tar         0.000512 ***
## nicotine    0.492035
## ---
## Signif. codes:
##   0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1  ' ' 1
##
```

```
## Residual standard error: 1.413 on 22 degrees of freedom
## Multiple R-squared:  0.9186, Adjusted R-squared:  0.9112
## F-statistic: 124.1 on 2 and 22 DF,  p-value: 1.042e-12
```

R-squared has dropped from 0.9186 to . . . 0.9186! That is, taking out `weight` has not just had a minimal effect on R-squared; it's not changed R-squared at all. This is because `weight` was so far from being significant: it literally had *nothing* to add.

Another way of achieving the same thing is via the function `update`, which takes a fitted model object and describes the *change* that you want to make:

```
cigs.2a = update(cigs.1, . ~ . - weight)
summary(cigs.2a)
```

```
##
## Call:
## lm(formula = co ~ tar + nicotine, data = cigs)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -2.89941 -0.78470 -0.00144  0.91585  2.43064
##
## Coefficients:
##             Estimate Std. Error t value
## (Intercept)   3.0896     0.8438   3.662
## tar           0.9625     0.2367   4.067
## nicotine     -2.6463     3.7872  -0.699
##             Pr(>|t|)
## (Intercept) 0.001371 **
## tar         0.000512 ***
## nicotine    0.492035
## ---
## Signif. codes:
##   0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1  ' ' 1
##
## Residual standard error: 1.413 on 22 degrees of freedom
## Multiple R-squared:  0.9186, Adjusted R-squared:  0.9112
## F-statistic: 124.1 on 2 and 22 DF,  p-value: 1.042e-12
```

This can be shorter than describing the whole model again, as you do with the `cigs.2` version of `lm`. The syntax is that you first specify a "base" fitted model object that you're going to update. Because the model `cigs.1` contains all the information about the kind of model it is, and which data frame the data come from, R already knows that this is a linear multiple regression and which

x's it contains. The second thing to describe is the change from the "base". In this case, we want to use the same response variable and all the same explanatory variables that we had before, except for `weight`. This is specified by a special kind of model formula where `.` means "whatever was there before": in English, "same response and same explanatories except take out `weight`".

(d) Fit a regression predicting carbon monoxide from `nicotine` *only*, and display the summary.

Solution
As you would guess:

```
cigs.3 = lm(co ~ nicotine, data = cigs)
summary(cigs.3)
```

```
##
## Call:
## lm(formula = co ~ nicotine, data = cigs)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -3.3273 -1.2228  0.2304  1.2700  3.9357
##
## Coefficients:
##             Estimate Std. Error t value
## (Intercept)   1.6647     0.9936   1.675
## nicotine     12.3954     1.0542  11.759
##             Pr(>|t|)
## (Intercept)    0.107
## nicotine    3.31e-11 ***
## ---
## Signif. codes:
##   0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1  ' ' 1
##
## Residual standard error: 1.828 on 23 degrees of freedom
## Multiple R-squared:  0.8574, Adjusted R-squared:  0.8512
## F-statistic: 138.3 on 1 and 23 DF,  p-value: 3.312e-11
```

(e) `nicotine` was far from being significant in the model of (c), and yet in the model of (d), it was *strongly* significant, and the R-squared value of (d) was almost as high as that of (c). What does this say about the importance of `nicotine` as an explanatory variable? Explain, as briefly as you can manage.

Solution

What this says is that you *cannot* say anything about the "importance" of `nicotine` without also describing the context that you're talking about. *By itself*, `nicotine` is important, but *when you have 'tar' in the model*, `nicotine` is not important: precisely, it now has nothing to add over and above the predictive value that `tar` has. You might guess that this is because `tar` and `nicotine` are "saying the same thing" in some fashion. We'll explore that in a moment.

(f) Make a "pairs plot": that is, scatter plots between all pairs of variables. This can be done by feeding the whole data frame into `plot`. Do you see any strong relationships that do *not* include `co`? Does that shed any light on the last part? Explain briefly (or "at length" if that's how it comes out).

This is a base graphics graph rather than a ggplot one, but it will do for our purposes.

Solution

Plot the entire data frame:

```
plot(cigs)
```

We're supposed to ignore `co`, but I comment that strong relationships between `co` and *both* of `tar` and `nicotine` show up here, along with `weight` being at most weakly related to anything else.

That leaves the relationship of `tar` and `nicotine` with each other. That also looks like a strong linear trend. When you have correlations between explanatory variables, it is called "multicollinearity".

I mentioned a while back (in class) that having correlated x's was trouble. Here is where we find out why. The problem is that when `co` is large, `nicotine` is large, and a large value of `tar` will come along with it. So we don't know whether a large value of `co` is caused by a large value of `tar` or a large value of `nicotine`: there is no way to separate out their effects because in effect they are "glued together".

You might know of this effect (in an experimental design context) as "confounding": the effect of `tar` on `co` is confounded with the

effect of `nicotine` on `co`, and you can't tell which one deserves the credit for predicting `co`.

If you were able to design an experiment here, you could (in principle) manufacture a bunch of cigarettes with high tar; some of them would have high nicotine and some would have low. Likewise for low tar. Then the correlation between `nicotine` and `tar` would go away, their effects on `co` would no longer be confounded, and you could see unambiguously which one of the variables deserves credit for predicting `co`. Or maybe it depends on both, genuinely, but at least then you'd know.

We, however, have an observational study, so we have to make do with the data we have. Confounding is one of the risks we take when we work with observational data.

This was a "base graphics" plot. There is a way of doing a `ggplot`-style "pairs plot", as this is called, thus:

```
library(GGally)

##
## Attaching package: 'GGally'

## The following object is masked from 'package:dplyr':
##
##     nasa

cigs %>% ggpairs()
```

As ever, `install.packages` first, in the likely event that you don't have this package installed yet. Once you do, though, I think this is a nicer way to get a pairs plot.

This plot is a bit more sophisticated: instead of just having the scatterplots of the pairs of variables in the row and column, it uses

the diagonal to show a "kernel density" (a smoothed-out histogram), and upper-right it shows the correlation between each pair of variables. The three correlations between `co`, `tar` and `nicotine` are clearly the highest.

If you want only some of the columns to appear in your pairs plot, `select` them first, and then pass that data frame into `ggpairs`. Here, we found that `weight` was not correlated with anything much, so we can take it out and then make a pairs plot of the other variables:

```
cigs %>% select(-weight) %>% ggpairs()
```

The three correlations that remain are all very high, which is entirely consistent with the strong linear relationships that you see bottom left.

13.3 *Maximal oxygen uptake in young boys*

A physiologist wanted to understand the relationship between physical characteristics of pre-adolescent boys and their maximal oxygen uptake (millilitres of oxygen per kilogram of body weight). The data are in link for a random sample of 10 pre-adolescent boys. The variables are (with units):

- `uptake`: Oxygen uptake (millitres of oxygen per kilogram of body weight)
- `age`: boy's age (years)
- `height`: boy's height (cm)
- `weight`: boy's weight (kg)
- `chest`: chest depth (cm).

(a) Read the data into R and confirm that you do indeed have 10 observations.

Solution

```
my_url = "http://www.utsc.utoronto.ca/~butler/c32/youngboys.txt"
boys = read_delim(my_url, " ")
```

```
## Parsed with column specification:
## cols(
##   uptake = col_double(),
##   age = col_double(),
##   height = col_double(),
##   weight = col_double(),
##   chest = col_double()
## )
```

```
boys
```

```
## # A tibble: 10 x 5
##    uptake   age height weight chest
##     <dbl> <dbl>  <dbl>  <dbl> <dbl>
##  1   1.54   8.4   132    29.1  14.4
##  2   1.74   8.7   136.   29.7  14.5
##  3   1.32   8.9   128.   28.4  14
##  4   1.5    9.9   131.   28.8  14.2
##  5   1.46   9     130    25.9  13.6
##  6   1.35   7.7   128.   27.6  13.9
##  7   1.53   7.3   130.   29    14
##  8   1.71   9.9   138.   33.6  14.6
##  9   1.27   9.3   127.   27.7  13.9
## 10   1.5    8.1   132.   30.8  14.5
```

10 boys (rows) indeed.

(b) Fit a regression predicting oxygen uptake from all the other variables, and display the results.

Solution

Fitting four explanatory variables with only ten observations is likely to be pretty shaky, but we press ahead regardless:

```
boys.1 = lm(uptake ~ age + height + weight + chest,
    data = boys)
summary(boys.1)
```

```
##
```

```
## Call:
## lm(formula = uptake ~ age + height + weight + chest, data = boys)
##
## Residuals:
##         1         2         3         4
## -0.020697  0.019741 -0.003649  0.038470
##         5         6         7         8
## -0.023639 -0.026026  0.050459 -0.014380
##         9        10
##  0.004294 -0.024573
##
## Coefficients:
##              Estimate Std. Error t value
## (Intercept) -4.774739   0.862818  -5.534
## age         -0.035214   0.015386  -2.289
## height       0.051637   0.006215   8.308
## weight      -0.023417   0.013428  -1.744
## chest        0.034489   0.085239   0.405
##             Pr(>|t|)
## (Intercept) 0.002643 **
## age         0.070769 .
## height      0.000413 ***
## weight      0.141640
## chest       0.702490
## ---
## Signif. codes:
##   0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1  ' ' 1
##
## Residual standard error: 0.03721 on 5 degrees of freedom
## Multiple R-squared:  0.9675, Adjusted R-squared:  0.9415
## F-statistic:  37.2 on 4 and 5 DF,  p-value: 0.0006513
```

(c) (A one-mark question.) Would you say, on the evidence so far, that the regression fits well or badly? Explain (very) briefly.

Solution

R-squared of 0.97 (97%) is very high, so I'd say this regression fits very well. That's all. I said "on the evidence so far" to dissuade you from overthinking this, or thinking that you needed to produce some more evidence. That, plus the fact that this was only one mark.

(d) It seems reasonable that an older boy should have a greater oxygen uptake, all else being equal. Is this supported by your output? Explain briefly.

Solution

If an older boy has greater oxygen uptake (the "all else equal" was a hint), the slope of `age` should be positive. It is not: it is -0.035, so it is suggesting (all else equal) that a greater age goes with a *smaller* oxygen uptake. The reason why this happens (which you didn't need, but you can include it if you like) is that `age` has a non-small P-value of 0.07, so that the `age` slope is not significantly different from zero. With all the other variables, `age` has nothing to *add* over and above them, and we could therefore remove it.

(e) It seems reasonable that a boy with larger weight should have larger lungs and thus a *statistically significantly* larger oxygen uptake. Is that what happens here? Explain briefly.

Solution

Look at the P-value for `weight`. This is 0.14, not small, and so a boy with larger weight does not have a significantly larger oxygen uptake, all else equal. (The slope for `weight` is not significantly different from zero either.) I emphasized "statistically significant" to remind you that this means to do a test and get a P-value.

(f) Fit a model that contains only the significant explanatory variables from your first regression. How do the R-squared values from the two regressions compare? (The last sentence asks for more or less the same thing as the next part. Answer it either here or there. Either place is good.)

Solution

Only `height` is significant, so that's the only explanatory variable we need to keep. I would just do the regression straight rather than using `update` here:

```
boys.2 = lm(uptake ~ height, data = boys)
summary(boys.2)
```

```
##
## Call:
## lm(formula = uptake ~ height, data = boys)
##
## Residuals:
##       Min        1Q    Median        3Q
## -0.069879 -0.033144  0.001407  0.009581
##       Max
##  0.084012
##
## Coefficients:
##              Estimate Std. Error t value
```

```
## (Intercept) -3.843326   0.609198  -6.309
## height       0.040718   0.004648   8.761
##             Pr(>|t|)
## (Intercept) 0.000231 ***
## height      2.26e-05 ***
## ---
## Signif. codes:
##   0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1  ' ' 1
##
## Residual standard error: 0.05013 on 8 degrees of freedom
## Multiple R-squared:  0.9056, Adjusted R-squared:  0.8938
## F-statistic: 76.75 on 1 and 8 DF,  p-value: 2.258e-05
```

If you want, you can use `update` here, which looks like this:

```
boys.2a = update(boys.1, . ~ . - age - weight -
    chest)
summary(boys.2a)
```

```
##
## Call:
## lm(formula = uptake ~ height, data = boys)
##
## Residuals:
##       Min        1Q    Median        3Q
## -0.069879 -0.033144  0.001407  0.009581
##       Max
##  0.084012
##
## Coefficients:
##              Estimate Std. Error t value
## (Intercept) -3.843326   0.609198  -6.309
## height       0.040718   0.004648   8.761
##             Pr(>|t|)
## (Intercept) 0.000231 ***
## height      2.26e-05 ***
## ---
## Signif. codes:
##   0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1  ' ' 1
##
## Residual standard error: 0.05013 on 8 degrees of freedom
## Multiple R-squared:  0.9056, Adjusted R-squared:  0.8938
## F-statistic: 76.75 on 1 and 8 DF,  p-value: 2.258e-05
```

This doesn't go quite so smoothly here because there are three variables being removed, and it's a bit of work to type them all.

(g) How has R-squared changed between your two regressions? Describe what you see in a few words.

Solution

R-squared has dropped by a bit, from 97% to 91%. (Make your own call: pull out the two R-squared numbers, and say a word or two about how they compare. I don't much mind what you say: "R-squared has decreased (noticeably)", "R-squared has hardly changed". But say something.)

(h) Carry out a test comparing the fit of your two regression models. What do you conclude, and therefore what recommendation would you make about the regression that would be preferred?

Solution

The word "test" again implies something that produces a P-value with a null hypothesis that you might reject. In this case, the test that compares two models differing by more than one x uses `anova`, testing the null hypothesis that the two regressions are equally good, against the alternative that the bigger (first) one is better. Feed `anova` two fitted model objects, smaller first:

```
anova(boys.2, boys.1)
```

```
## Analysis of Variance Table
##
## Model 1: uptake ~ height
## Model 2: uptake ~ age + height + weight + chest
##   Res.Df       RSS Df Sum of Sq      F
## 1      8 0.0201016
## 2      5 0.0069226  3  0.013179 3.1729
##   Pr(>F)
## 1
## 2  0.123
```

This P-value of 0.123 is not small, so we do not reject the null hypothesis. There is not a significant difference in fit between the two models. Therefore, we should go with the smaller model `boys.2` because it is simpler.

That drop in R-squared from 97% to 91% was, it turns out, *not* significant: the three extra variables could have produced a change in R-squared like that, *even if they were worthless*.

Recall that adding x's to a regression will always make R-squared go up, even if they are just random noise.

If you have learned about "adjusted R-squared", you might recall that this is supposed to go down *only* if the variables you took out should not have been taken out. But adjusted R-squared goes down here as well, from 94% to 89% (not quite as much, therefore). What

happens is that adjusted R-squared is rather more relaxed about keeping variables than the `anova` F-test is; if we had used an α of something like 0.10, the decision between the two models would have been a lot closer, and this is reflected in the adjusted R-squared values.

(i) Obtain a table of correlations between all the variables in the data frame. Do this by feeding the whole data frame into `cor`. We found that a regression predicting oxygen uptake from just `height` was acceptably good. What does your table of correlations say about why that is? (Hint: look for all the correlations that are *large*.)

Solution
Correlations first:

```
cor(boys)
```

```
##           uptake       age    height
## uptake 1.0000000 0.1361907 0.9516347
## age    0.1361907 1.0000000 0.3274830
## height 0.9516347 0.3274830 1.0000000
## weight 0.6576883 0.2307403 0.7898252
## chest  0.7182659 0.1657523 0.7909452
##           weight     chest
## uptake 0.6576883 0.7182659
## age    0.2307403 0.1657523
## height 0.7898252 0.7909452
## weight 1.0000000 0.8809605
## chest  0.8809605 1.0000000
```

The correlations with `age` are all on the low side, but all the other correlations are high, not just between `uptake` and the other variables, but between the explanatory variables as well.

Why is this helpful in understanding what's going on? Well, imagine a boy with large height (a tall one). The regression `boys.2` says that this alone is enough to predict that such a boy's oxygen uptake is likely to be large, since the slope is positive. But the correlations tell you more: a boy with large height is also (somewhat) likely to be older (have large age), heavier (large weight) and to have larger `chest` cavity. So oxygen uptake does depend on those other variables as well, but once you know `height` you can make a good guess at their values; you don't need to know them.

Further remarks: `age` has a low correlation with `uptake`, so its non-significance earlier appears to be "real": it really does have nothing extra to say, because the other variables have a stronger link with

uptake than age. Height, however, seems to be the best way of relating oxygen uptake to any of the other variables. I think the suppositions from earlier about relating oxygen uptake to "bigness" in some sense are actually sound, but age and weight and chest capture "bigness" worse than height does. Later, when you learn about Principal Components, you will see that the first principal component, the one that best captures how the variables vary together, is often "bigness" in some sense.

This is not, I don't think, a real word, but I mean size emphasizing how big a boy is generally, rather than how small.

Another way to think about these things is via pairwise scatterplots. The nicest way to produce these is via ggpairs from package GGally:

```
boys %>% ggpairs()
```

A final remark: with five variables, we really ought to have more than ten observations (something like 50 would be better). But with more observations and the same correlation structure, the same issues would come up again, so the question would not be materially changed.

13.4 *Facebook friends and grey matter*

Is there a relationship between the number of Facebook friends a person has, and the density of grey matter in the areas of the brain associated with social perception and associative memory? To find out, a 2012 study measured both of these variables for a sample of 40 students at City University in London (England). The data are at link. The grey matter density is on a z-score standardized scale. The values are separated by *tabs*.

The aim of this question is to produce an R Markdown report that contains your answers to the questions below.

You should aim to make your report flow smoothly, so that it would be pleasant for a grader to read, and can stand on its own as an analysis (rather than just being the answer to a question that I set you). Some suggestions: give your report a title and arrange it into sections with an Introduction; add a small amount of additional text here and there explaining what you are doing and why. I don't expect you to spend a large amount of time on this, but I *do* hope you will make some effort. (My report came out to 4 Word pages.)

(a) Read in the data and make a scatterplot for predicting the number of Facebook friends from the grey matter density. On your scatterplot, add a smooth trend.

Solution

Begin your document with a code chunk containing `library(tidyverse)`. The data values are separated by tabs, which you will need to take into account:

```
my_url = "http://www.utsc.utoronto.ca/~butler/c32/facebook.txt"
fb = read_tsv(my_url)
```

```
## Parsed with column specification:
## cols(
##   GMdensity = col_double(),
##   FBfriends = col_integer()
## )
```

```
fb
```

```
## # A tibble: 40 x 2
##    GMdensity FBfriends
##        <dbl>     <int>
##  1      -1.8        23
##  2       0.1        35
##  3      -1.2        80
##  4      -0.4       110
##  5      -0.9       120
##  6      -2.1       140
##  7      -1.5       168
##  8       0.5       132
##  9       0.6       154
## 10      -0.5       241
## # ... with 30 more rows
```

```
ggplot(fb, aes(x = GMdensity, y = FBfriends)) +
    geom_point() + geom_smooth()
```

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

(b) Describe what you see on your scatterplot: is there a trend, and if so, what kind of trend is it? (Don't get too taken in by the exact shape of your smooth trend.) Think "form, direction, strength".

Solution

I'd say there seems to be a weak, upward, apparently linear trend. The points are not especially close to the trend, so I don't think there's any justification for calling this other than "weak". (If you think the trend is, let's say, "moderate", you ought to say what makes you think that: for example, that the people with a lot of Facebook friends also tend to have a higher grey matter density. I can live with a reasonably-justified "moderate".) The reason I said not to get taken in by the shape of the smooth trend is that this has a "wiggle" in it: it goes down again briefly in the middle. But this is likely a quirk of the data, and the trend, if there is any, seems to be an upward one.

(c) Fit a regression predicting the number of Facebook friends from the grey matter density, and display the output.

Solution

That looks like this. You can call the "fitted model object" whatever you like, but you'll need to get the capitalization of the variable names correct:

```
fb.1 = lm(FBfriends ~ GMdensity, data = fb)
summary(fb.1)
```

```
##
## Call:
## lm(formula = FBfriends ~ GMdensity, data = fb)
```

```
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -339.89 -110.01   -5.12   99.80  303.64
##
## Coefficients:
##             Estimate Std. Error t value
## (Intercept)   366.64      26.35  13.916
## GMdensity      82.45      27.58   2.989
##             Pr(>|t|)
## (Intercept)  < 2e-16 ***
## GMdensity    0.00488 **
## ---
## Signif. codes:
##   0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1  ' ' 1
##
## Residual standard error: 165.7 on 38 degrees of freedom
## Multiple R-squared:  0.1904, Adjusted R-squared:  0.1691
## F-statistic: 8.936 on 1 and 38 DF,  p-value: 0.004882
```

I observe, though I didn't ask you to, that the R-squared is pretty awful, going with a correlation of

```
sqrt(0.1904)
```

```
## [1] 0.4363485
```

which *would* look like as weak of a trend as we saw.

Correlations have to go up beyond 0.50 before they start looking at all interesting.

(d) Is the slope of your regression line significantly different from zero? What does that mean, in the context of the data?

Solution

The P-value of the slope is 0.005, which is less than 0.05. Therefore the slope *is* significantly different from zero. That means that the number of Facebook friends really does depend on the grey matter density, for the whole population of interest and not just the 40 students observed here (that were a sample from that population). I don't mind so much what you think the population is, but it needs to be clear that the relationship applies to a population. Another way to approach this is to say that you would expect this relationship to show up again in another similar experiment. That also works, because it gets at the idea of reproducibility.

(e) Are you surprised by the results of parts (b) and (d)? Explain briefly.

Solution

I *am* surprised, because I thought the trend on the scatterplot was so weak that there would not be a significant slope. I guess there was enough of an upward trend to be significant, and with $n = 40$ observations we were able to get a significant slope out of that scatterplot. With this many observations, even a weak correlation can be significantly nonzero. You can be surprised or not, but you need to have some kind of consideration of the strength of the trend on the scatterplot as against the significance of the slope. For example, if you decided that the trend was "moderate" in strength, you would be justified in being less surprised than I was. Here, there is the usual issue that we have proved that the slope is not zero (that the relationship is not flat), but we may not have a very clear idea of what the slope actually *is*. There are a couple of ways to get a confidence interval. The obvious one is to use R as a calculator and go up and down twice its standard error (to get a rough idea):

```
82.45 + 2 * 27.58 * c(-1, 1)
```

```
## [1]  27.29 137.61
```

The `c()` thing is to get both confidence limits at once. The smoother way is this:

```
confint(fb.1)
```

```
##                 2.5 %   97.5 %
## (Intercept) 313.30872 419.9810
## GMdensity    26.61391 138.2836
```

Feed `confint` a "fitted model object" and it'll give you confidence intervals (by default 95%) for all the parameters in it.

The confidence interval for the slope goes from about 27 to about 138. That is to say, a one-unit increase in grey matter density goes with an increase in Facebook friends of this much. This is not especially insightful: it's bigger than zero (the test was significant), but other than that, it could be almost anything. *This* is where the weakness of the trend comes back to bite us. With this much scatter in our data, we need a *much* larger sample size to estimate accurately how big an effect grey matter density has.

(f) Obtain a scatterplot with the regression line on it.

Solution

Just a modification of (a):

```
ggplot(fb, aes(x = GMdensity, y = FBfriends)) +
    geom_point() + geom_smooth(method = "lm")
```

(g) Obtain a plot of the residuals from the regression against the fitted values, and comment briefly on it.

Solution

This is, to my mind, the easiest way:

```
ggplot(fb.1, aes(x = .fitted, y = .resid)) + geom_point()
```

There is some "magic" here, since the fitted model object is not actually a data frame, but it works this way. That looks to me like a completely random scatter of points. Thus, I am completely happy with the straight-line regression that we fitted, and I see no need to improve it.

(You should make two points here: one, describe what you see, and two, what it implies about whether or not your regression is satisfactory.)

Compare that residual plot with this one:

```
ggplot(fb.1, aes(x = .fitted, y = .resid)) + geom_point() +
    geom_smooth()
```

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

Now, why did I try adding a smooth trend, and why is it not necessarily a good idea? The idea of a residual plot is that there should be no trend, and so the smooth trend curve ought to go straight across. The problem is that it will tend to wiggle, just by chance, as here: it looks as if it goes up and down before flattening out. But if you look at the points, *they* are all over the place, not close to the smooth trend at all. So the smooth trend is rather deceiving. Or, to put it another way, to indicate a real problem, the smooth trend would have to be a *lot* farther from flat than this one is. I'd call this one basically flat.

13.5 *Endogenous nitrogen excretion in carp*

A paper in Fisheries Science reported on variables that affect "endogenous nitrogen excretion" or ENE in carp raised in Japan. A number of carp were divided into groups based on body weight, and each group was placed in a different tank. The mean body weight of the carp placed in each tank was recorded. The carp were then fed a protein-free diet three times daily for a period of 20 days. At the end of the experiment, the amount of ENE in each tank was measured, in milligrams of total fish body weight per day. (Thus it should not matter that some of the tanks had more fish than others, because the scaling is done properly.)

For this question, write a report in R Markdown that answers the questions below and contains some narrative that describes your analysis. Create an HTML document from your R Markdown.

(a) Read the data in from link. There are 10 tanks.

Solution

Just this. Listing the data is up to you, but doing so and commenting that the values appear to be correct will improve your report.

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

```
## Parsed with column specification:
## cols(
##   tank = col_integer(),
##   bodyweight = col_double(),
##   ENE = col_double()
## )
```

```
carp
```

```
## # A tibble: 10 x 3
##     tank bodyweight   ENE
##    <int>      <dbl> <dbl>
##  1     1       11.7  15.3
##  2     2       25.3   9.3
##  3     3       90.2   6.5
##  4     4      213     6
##  5     5       10.2  15.7
##  6     6       17.6  10
##  7     7       32.6   8.6
##  8     8       81.3   6.4
##  9     9      142.    5.6
## 10    10      286.    6
```

(b) Create a scatterplot of ENE (response) against bodyweight (explanatory). Add a smooth trend to your plot.

Solution

```
ggplot(carp, aes(x = bodyweight, y = ENE)) + geom_point() +
    geom_smooth()
```

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

This part is just about getting the plot. Comments are coming in a minute. Note that `ENE` is capital letters, so that `ene` will not work.

(c) Is there an upward or downward trend (or neither)? Is the relationship a line or a curve? Explain briefly.

Solution

The trend is downward: as bodyweight increases, ENE decreases. However, the decrease is rapid at first and then levels off, so the relationship is nonlinear. I want some kind of support for an assertion of non-linearity: anything that says that the slope or rate of decrease is not constant is good.

(d) Fit a straight line to the data, and obtain the R-squared for the regression.

Solution

`lm`. The first stage is to fit the straight line, saving the result in a variable, and the second stage is to look at the "fitted model object", here via `summary`:

```
carp.1 = lm(ENE ~ bodyweight, data = carp)
summary(carp.1)
```

```
##
## Call:
## lm(formula = ENE ~ bodyweight, data = carp)
##
## Residuals:
##    Min     1Q Median     3Q    Max
## -2.800 -1.957 -1.173  1.847  4.572
##
```

```
## Coefficients:
##             Estimate Std. Error t value
## (Intercept) 11.40393    1.31464   8.675
## bodyweight  -0.02710    0.01027  -2.640
##             Pr(>|t|)
## (Intercept) 2.43e-05 ***
## bodyweight    0.0297 *
## ---
## Signif. codes:
##   0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1  ' ' 1
##
## Residual standard error: 2.928 on 8 degrees of freedom
## Multiple R-squared:  0.4656, Adjusted R-squared:  0.3988
## F-statistic: 6.971 on 1 and 8 DF,  p-value: 0.0297
```

Finally, you need to give me a (suitably rounded) value for R-squared: 46.6% or 47% or the equivalents as a decimal. I just need the value at this point. This kind of R-squared is actually pretty good for natural data, but the issue is whether we can improve it by fitting a non-linear model.

The suspicion being that we can, since the scatterplot suggested serious non-linearity.

(e) Obtain a residual plot (residuals against fitted values) for this regression. Do you see any problems? If so, what does that tell you about the relationship in the data?

Solution

This is the easiest way: feed the output of the regression straight into `ggplot`:

```
ggplot(carp.1, aes(x = .fitted, y = .resid)) +
    geom_point()
```

(f) Fit a parabola to the data (that is, including an x-squared term). Compare the R-squared values for the models in this part and part (d). Does that suggest that the parabola model is an improvement here over the linear model?

Solution

Add bodyweight-squared to the regression. Don't forget the `I()`:

```
carp.2 = lm(ENE ~ bodyweight + I(bodyweight^2),
    data = carp)
summary(carp.2)
```

```
##
## Call:
## lm(formula = ENE ~ bodyweight + I(bodyweight^2), data = carp)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -2.0834 -1.7388 -0.5464  1.3841  2.9976
##
## Coefficients:
##                   Estimate Std. Error
## (Intercept)     13.7127373  1.3062494
## bodyweight      -0.1018390  0.0288109
## I(bodyweight^2)  0.0002735  0.0001016
##                 t value Pr(>|t|)
## (Intercept)      10.498 1.55e-05 ***
## bodyweight       -3.535  0.00954 **
## I(bodyweight^2)   2.692  0.03101 *
## ---
## Signif. codes:
##   0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1  ' ' 1
##
## Residual standard error: 2.194 on 7 degrees of freedom
## Multiple R-squared:  0.7374, Adjusted R-squared:  0.6624
## F-statistic: 9.829 on 2 and 7 DF,  p-value: 0.009277
```

R-squared has gone up from 47% to 74%, a substantial improvement. This suggests to me that the parabola model is a substantial improvement.

Again, not a surprise, given our initial scatterplot.

I try to avoid using the word "significant" in this context, since we haven't actually done a test of significance.

The reason for the `I()` is that the up-arrow has a special meaning in `lm`, relating to interactions between factors (as in ANOVA), that we don't want here. Putting `I()` around it means "use as is", that is,

raise bodyweight to power 2, rather than using the special meaning of the up-arrow in `lm`.

Because it's the up-arrow that is the problem, this applies whenever you're raising an explanatory variable to a power (or taking a reciprocal or a square root, say).

(g) Is the test for the slope coefficient for the squared term significant? What does this mean?

Solution

Look along the `bodyweight`-squared line to get a P-value of 0.031. This is less than the default 0.05, so it *is* significant. This means, in short, that the quadratic model is a significant *improvement* over the linear one. Said longer: the null hypothesis being tested is that the slope coefficient of the squared term is zero (that is, that the squared term has nothing to add over the linear model). This is rejected, so the squared term has *something* to add in terms of quality of prediction.

Now we can use that word *significant*.

(h) Make the scatterplot of part (b), but add the fitted curve. Describe any way in which the curve fails to fit well.

Solution

This is a bit slippery, because the points to plot and the fitted curve are from different data frames. What you do in this case is to put a `data=` in one of the `geoms`, which says "don't use the data frame that was in the `ggplot`, but use this one instead". I would think about starting with the regression object `carp.2` as my base data frame, since we want (or I want) to do two things with that: plot the fitted values and join them with lines. Then I want to add the original data, just the points:

```
ggplot(carp.2, aes(x = carp$bodyweight, y = .fitted),
    colour = "blue") + geom_line(colour = "blue") +
    geom_point(data = carp, aes(x = bodyweight,
        y = ENE))
```

This works, but is not very aesthetic, because the bodyweight that is plotted against the fitted values is in the wrong data frame, and so we have to use the dollar-sign thing to get it from the right one.

A better way around this is "augment" the data with output from the regression object. This is done using `augment` from package `broom`:

```
library(broom)
carp.2a = augment(carp.2, carp)
carp.2a
```

```
## # A tibble: 10 x 10
##     tank bodyweight   ENE .fitted .se.fit
##  * <int>      <dbl> <dbl>   <dbl>   <dbl>
##  1     1       11.7  15.3   12.6    1.07
##  2     2       25.3   9.3   11.3    0.886
##  3     3       90.2   6.5    6.75   1.07
##  4     4      213     6      4.43   1.25
##  5     5       10.2  15.7   12.7    1.10
##  6     6       17.6  10     12.0    0.980
##  7     7       32.6   8.6   10.7    0.828
##  8     8       81.3   6.4    7.24   1.01
##  9     9      142.    5.6    4.78   1.31
## 10    10      286.    6      6.94   2.05
## # ... with 5 more variables: .resid <dbl>,
## #   .hat <dbl>, .sigma <dbl>, .cooksd <dbl>,
## #   .std.resid <dbl>
```

so now you see what `carp.2a` has in it, and then:

```
g = ggplot(carp.2a, aes(x = bodyweight, y = .fitted)) +
    geom_line(colour = "blue") + geom_point(aes(y = ENE))
```

This is easier coding: there are only two non-standard things. The first is that the fitted-value lines should be a distinct colour like blue so that you can tell them from the data points. The second thing is that for the second `geom_point`, the one that plots the data, the x coordinate `bodyweight` is correct so that we don't have to change that; we only have to change the y-coordinate, which is `ENE`. The plot is this:

```
g
```

Concerning interpretation, you have a number of possibilities here. The simplest is that the points in the middle are above the curve, and the points at the ends are below. (That is, negative residuals at the ends, and positive ones in the middle, which gives you a hint for the next part.) Another is that the parabola curve fails to capture the *shape* of the relationship; for example, I see nothing much in the data suggesting that the relationship should go back up, and even given that, the fitted curve doesn't go especially near any of the points.

I was thinking that the data should be fit better by something like the left half of an upward-opening parabola, but I guess the curvature on the left half of the plot suggests that it needs most of the left half of the parabola just to cover the left half of the plot.

The moral of the story, as we see in the next part, is that the parabola is the wrong curve for the job.

(i) Obtain a residual plot for the parabola model. Do you see any problems with it? (If you do, I'm not asking you to do anything about them in this question, but I will.)

Solution

The same idea as before for the other residual plot. Use the fitted model object `carp.2` as your data frame for the `ggplot`:

```
ggplot(carp.2, aes(x = .fitted, y = .resid)) +
    geom_point()
```

I think this is *still* a curve (or, it goes down and then sharply up at the end). Either way, there is still a pattern.

That was all I needed, but as to what this means: our parabola was a curve all right, but it appears not to be the right *kind* of curve. I think the original data looks more like a hyperbola (a curve like $y = 1/x$) than a parabola, in that it seems to decrease fast and then gradually to a limit, and *that* suggests, as in the class example, that we should try an asymptote model. Note how I specify it, with the `I()` thing again, since `/` has a special meaning to `lm` in the same way that ˆ does:

```
carp.3 = lm(ENE ~ I(1/bodyweight), data = carp)
summary(carp.3)
```

```
##
## Call:
## lm(formula = ENE ~ I(1/bodyweight), data = carp)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -1.29801 -0.12830  0.04029  0.26702  0.91707
##
## Coefficients:
##                 Estimate Std. Error t value
## (Intercept)       5.1804     0.2823   18.35
## I(1/bodyweight) 107.6690     5.8860   18.29
##                 Pr(>|t|)
## (Intercept)     8.01e-08 ***
## I(1/bodyweight) 8.21e-08 ***
```

```
## ---
## Signif. codes:
##   0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1  ' ' 1
##
## Residual standard error: 0.6121 on 8 degrees of freedom
## Multiple R-squared:  0.9766, Adjusted R-squared:  0.9737
## F-statistic: 334.6 on 1 and 8 DF,  p-value: 8.205e-08
```

That fits *extraordinarily* well, with an R-squared up near 98%. The intercept is the asymptote, which suggests a (lower) limit of about 5.2 for `ENE` (in the limit for large bodyweight). We would have to ask the fisheries scientist whether this kind of thing is a reasonable biological mechanism. It says that a carp always has some ENE, no matter how big it gets, but a smaller carp will have a lot more.

Does the fitted value plot look reasonable now? This is `augment` again since the fitted values and observed data come from different data frames:

```
library(broom)
augment(carp.3, carp) %>% ggplot(aes(x = bodyweight,
    y = .fitted)) + geom_line(colour = "blue") +
    geom_point(aes(y = ENE))
```

I'd say that does a really nice job of fitting the data. But it would be nice to have a few more tanks with large-bodyweight fish, to convince us that we have the shape of the trend right.

And, as ever, the residual plot. That's a lot easier than the plot we just did:

```
ggplot(carp.3, aes(x = .fitted, y = .resid)) +
    geom_point()
```

All in all, that looks pretty good (and certainly a vast improvement over the ones you got before).

When you write up your report, you can make it flow better by writing it in a way that suggests that each thing was the obvious thing to do next: that is, that *you* would have thought to do it next, rather than me telling you what to do.

My report (as an R Markdown file) is at link. Download it, knit it, play with it.

13.6 *Sparrowhawks*

One of nature's patterns is the relationship between the percentage of adult birds in a colony that return from the previous year, and the number of new adults that join the colony. Data for 13 colonies of sparrowhawks can be found at link. The columns are the percentage of adults returning from the previous year, and the number of new adults that join the colony.

(a) Create a new R Markdown report, give it a suitable title, and ask for HTML output. Answer the questions that follow in your report. At any stage, you can Knit HTML to see how the report looks so far.

Solution

In R Studio, select File, New File, R Markdown. Fill in the Title, Author and leave the Default Output Format at HTML. You'll see a template report with the document info at the top. This is my document info:

```
---
title: "Sparrowhawks"
author: "Ken Butler"
date: "October 20, 2018"
output: html_document
---
```

This is known in the jargon as a "YAML block". Below that is the template R Markdown document, which you can delete now or later.

YAML stands for *Yet Another Markup Language*, but we're not using it in this course, other than as the top bit of an R Markdown document.

(b) Read in the data and display the first few values. Add some text saying how many rows of data there are.

Solution

Read the data into a data frame. In your report, add some text like "we read in the data", perhaps after a section heading like "The data". Then add a *code chunk* by selecting Chunks and Insert Chunk, or by pressing control-alt-I. So far you have something like this.

```
---
title: "Sparrowhawks"
author: "Ken Butler"
date: "October 20, 2018"
output: html_document
---

## The data

We begin by reading in the data.

```{r}

```

```

Inside the code chunk, that is, in the bit between the backtick characters, put R code, just as you would type it at the Console or put in an R notebook. In this case, that would be the following code, minus the message that comes out of `read_delim`:

```
library(tidyverse)
my_url = "http://www.utsc.utoronto.ca/~butler/c32/sparrowhawk.txt"
sparrowhawks = read_delim(my_url, " ")

Parsed with column specification:
```

```
## cols(
##   returning = col_integer(),
##   newadults = col_integer()
## )
```

```
sparrowhawks
```

For you, it looks like this:

```
---
title: "Sparrowhawks"
author: "Ken Butler"
date: "October 20, 2018"
output: html_document
---

## The data

We begin by reading in the data, and loading in our usual stuff (for
use later):

```{r}
library(tidyverse)
my_url="http://www.utsc.utoronto.ca/~butler/c32/sparrowhawk.txt"
sparrowhawks=read_delim(my_url," ")
sparrowhawks
```

There are xxx rows of data altogether.

```

We don't know how many rows of data there are yet, so I've left a "placeholder" for it, when we figure it out. The file is annoyingly called `sparrowhawk.txt`, singular. Sorry about that. If you knit this (click on "Knit HTML" next to the ball of wool, or press control-shift-K), it should run, and you'll see a viewer pop up with the HTML output. Now you can see how many rows there are, and you can go back and edit the R Markdown and put in 13 in place of xxx, and knit again. You might be worried about how hard R is working with all this knitting. Don't worry about that. R can take it. Mine looked like this:

```
my_url="http://www.utsc.utoronto.ca/~butler/c32/sparrowhawk.txt"
sparrowhawks=read_delim(my_url," ")
```

```
## Parsed with column specification:
## cols(
##   returning = col_integer(),
##   newadults = col_integer()
## )
```

```
sparrowhawks
```

```
## # A tibble: 13 x 2
##    returning newadults
##        <int>     <int>
##  1        74         5
##  2        66         6
##  3        81         8
##  4        52        11
##  5        73        12
##  6        62        15
##  7        52        16
##  8        45        17
##  9        62        18
## 10        46        18
## 11        60        19
## 12        46        20
## 13        38        20
```

There are 13 rows of data altogether, with a value of the number of returning and new adults for each one.

There is a better way of adding values that come from the output, which I mention here in case you are interested (if you are not, feel free to skip this). What you do is to make what is called an "inline code chunk". Where you want a number to appear in the text, you have some R Markdown that looks like this:

```
There are `r nrow(sparrowhawks)` rows altogether.
```

The piece inside the backticks is the letter `r`, a space, and then one line of R code. The one line of code will be run, and all of the stuff within the backticks will be replaced in the output by the result of running the R code, in this case the number 13. Typically, you are extracting a number from the data, like the number of rows or a mean of something. If it's a decimal number, it will come out with a lot of decimal places unless you explicitly `round` it. OK, let me try it: the data frame has 13 rows altogether. I didn't type that number; it was calculated from the data frame. Woo hoo!

(c) Create a new section entitled "Exploratory analysis", and create a scatterplot for predicting number of new adults from the percentage of returning adults. Describe what you see, adding some suitable text to your report.

Solution

The R code you add should look like this, with the results shown (when you knit the report again):

```
library(tidyverse)
```

```r
ggplot(sparrowhawks, aes(x = returning, y = newadults)) +
    geom_point() + geom_smooth()
```

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

The piece of report that I added looks like this:

```
## Exploratory analysis

We should look at a scatterplot of the data, to see what, if any,
relationship there is. We add a smooth trend, to guide the eye:

```{r}
ggplot(sparrowhawks,aes(x=returning,y=newadults))+
  geom_point()+geom_smooth()
```

This seems to be something of a downward trend, though it is not
completely convincing. That is, a larger percentage of returning
adults is associated with a *smaller* number of new adults. Is the
relationship straight? Well, the smooth trend has a strange
wiggle in the middle, but the trend overall does not seem
obviously curved.
```

Note (i) that you have to do nothing special to get the plot to appear, and (ii) that I put "smaller" in italics, and you see how.

(d) Obtain the correlation between the two variables. Is this consistent with the scatterplot? Explain briefly. (The R function you need is `cor`. You can feed it a data frame.)

Solution

The appropriate R code is this, in another code chunk:

```r
with(sparrowhawks, cor(newadults, returning))
```

```
## [1] -0.7484673
```

Or you can ask for the correlations of the whole data frame:

```r
cor(sparrowhawks)
```

```
##            returning  newadults
## returning  1.0000000 -0.7484673
## newadults -0.7484673  1.0000000
```

This latter is a "correlation matrix" with a correlation between each column and each other column. Obviously the correlation between a column and itself is 1, and that is *not* the one we want.

I added this to the report (still in the Exploratory Analysis section, since it seems to belong there):

```
What is the correlation between these two variables?

```{r}
with(sparrowhawks,cor(newadults,returning))
```

The correlation is about -0.75. This seems surprisingly close to -1 to me, since I
didn't think the trend was that strong, but it is no surprise that the correlation
is negative, since the trend is definitely downward rather than
upward.
```

(e) Obtain the regression line for predicting the number of new adults from the percentage of returning adults.

Solution

This R code, in another code chunk:

```r
newadults.1 = lm(newadults ~ returning, data = sparrowhawks)
summary(newadults.1)
```

```
##
## Call:
## lm(formula = newadults ~ returning, data = sparrowhawks)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -5.8687 -1.2532  0.0508  2.0508  5.3071
##
## Coefficients:
##             Estimate Std. Error t value
## (Intercept) 31.93426    4.83762   6.601
## returning   -0.30402    0.08122  -3.743
##             Pr(>|t|)
## (Intercept) 3.86e-05 ***
## returning    0.00325 **
## ---
## Signif. codes:
##   0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1  ' ' 1
```

```
##
## Residual standard error: 3.667 on 11 degrees of freedom
## Multiple R-squared:  0.5602, Adjusted R-squared:  0.5202
## F-statistic: 14.01 on 1 and 11 DF,  p-value: 0.003248
```

(f) What are the intercept and slope of your regression line? Is the slope significant? What does that mean, in the context of the data?

Solution

See the output in the previous part. That's what we need to talk about. I added this to the report. I thought we deserved a new section here:

```
## Regression analysis


The regression for predicting the number of new adults from the
percentage of returning adults is as shown:

```{r}
newadults.1=lm(newadults~returning,data=sparrowhawks)
summary(newadults.1)
```

The intercept is about 32 and the slope is about -0.3. This means that
each additional one percentage point in returning adults is associated
with a *decrease* of approximately 0.3 new adults.

The P-value attached to the slope is about 0.003. This means that the
slope is *not* zero, and that there is a significant association
between the number of returning adults and the number of new
adults. (This is despite the apparently weak trend, and the small
number, 13, of data points.)
```

(g) Create a scatterplot of the data with the regression line on it.

Solution

This code. Using `geom_smooth` with `method="lm"` will add the regression line to the plot:

```r
ggplot(sparrowhawks, aes(x = returning, y = newadults)) +
    geom_point() + geom_smooth(method = "lm")
```

I added a bit of text to the report, which I will show in a moment.

(h) For short-lived birds, the association between these two variables is positive: changes in weather and food supply cause the populations of new and returning birds to increase together. For long-lived territorial birds, however, the association is negative because returning birds claim their territories in the colony and do not leave room for new recruits. Which type of species is the sparrowhawk? Add a short Conclusions section to your report with discussion of this issue.

Solution

My addition to the report looks like this:

```
71
72 A plot of the data with the regression line superimposed is as shown:
73
74 ```{r}
75 ggplot(sparrowhawks,aes(x=returning,y=newadults))+
76   geom_point()+geom_smooth(method="lm")
77 ```
78
79 ### Conclusions
80
81
82 The association is negative, as seen from the scatterplot and
83 correlation, and also from the negative slope of the regression
84 line. This suggests that sparrowhawks are long-lived territorial
85 birds rather than short-lived birds. We would expect to observe
86 returning birds displaying territorial behaviour that would
87 discourage new adults from joining the colony.
88
```

I think that rounds off the report nicely.

13.7 *Salaries of social workers*

Another salary-prediction question: does the number of years of work experience that a social worker has help to predict their salary? Data for 50 social workers are in link.

(a) Read the data into R. Check that you have 50 observations on two variables. Also do something to check that the years of experience and annual salary figures look reasonable overall.

Solution

```
my_url = "http://www.utsc.utoronto.ca/~butler/c32/socwork.txt"
soc = read_delim(my_url, " ")
```

```
## Parsed with column specification:
## cols(
##   experience = col_integer(),
##   salary = col_integer()
## )
```

```
soc
```

```
## # A tibble: 50 x 2
##    experience salary
##         <int>  <int>
##  1          7  26075
##  2         28  79370
##  3         23  65726
##  4         18  41983
##  5         19  62308
##  6         15  41154
##  7         24  53610
##  8         13  33697
##  9          2  22444
## 10          8  32562
## # ... with 40 more rows
```

That checks that we have the right *number* of observations; to check that we have sensible *values*, something like `summary` is called for:

```
summary(soc)
```

```
##    experience        salary
##  Min.   : 1.00   Min.   :16105
##  1st Qu.:13.50   1st Qu.:36990
##  Median :20.00   Median :50948
##  Mean   :18.12   Mean   :50171
##  3rd Qu.:24.75   3rd Qu.:65204
##  Max.   :28.00   Max.   :99139
```

A person working in any field cannot have a negative number of years of experience, and cannot have more than about 40 years of

experience (or else they would have retired). Our experience numbers fit that. Salaries had better be five or six figures, and salaries for social workers are not generally all that high, so these figures look reasonable.

A rather more `tidyverse` way is this:

```
soc %>% summarize_all(c("min", "max"))
```

```
## # A tibble: 1 x 4
##   experience_min salary_min experience_max
##            <dbl>      <dbl>          <dbl>
## 1              1      16105             28
## # ... with 1 more variable: salary_max <dbl>
```

This gets the minimum and maximum of all the variables. I would have liked them arranged in a nice rectangle (`min` and `max` as rows, the variables as columns), but that's not how this comes out.

Here is another:

```
soc %>% map_df(~quantile(.))
```

```
## # A tibble: 5 x 2
##   experience salary
##        <dbl>  <dbl>
## 1        1   16105
## 2       13.5 36990.
## 3       20   50948.
## 4       24.8 65204.
## 5       28   99139
```

These are the five-number summaries of each variable. Normally, they come with percentiles attached:

```
quantile(soc$experience)
```

```
##    0%   25%   50%   75%  100%
##  1.00 13.50 20.00 24.75 28.00
```

but the percentiles get lost in the transition to a `tibble`, and I haven't found out how to get them back.

This almost works:

```
soc %>% map_df(~enframe(quantile(.)))
```

```
## # A tibble: 10 x 2
##    name    value
##    <chr>   <dbl>
##  1 0%        1
```

```
##  2 25%      13.5
##  3 50%      20
##  4 75%      24.8
##  5 100%     28
##  6 0%    16105
##  7 25%   36990.
##  8 50%   50948.
##  9 75%   65204.
## 10 100%  99139
```

but, though we now have the percentiles, we've lost the names of the variables, so it isn't much better.

In this context, `map` says "do whatever is in the brackets for each column of the data frame". (That's the implied "for each".) The output from `quantile` is a vector that we would like to have display as a data frame, so `map_df` instead of any other form of `map`.

As you know, the `map` family is actually very flexible: they run a function "for each" anything and glue the results together, like this:

```
soc %>% map_dbl(median)
```

```
## experience     salary
##       20.0    50947.5
```

which gets the median for each variable. That's the same thing as this:

```
soc %>% summarize_all("median")
```

```
## # A tibble: 1 x 2
##   experience salary
##        <dbl>  <dbl>
## 1         20 50948.
```

(b) Make a scatterplot showing how salary depends on experience. Does the nature of the trend make sense?

Solution

The usual:

```
ggplot(soc, aes(x = experience, y = salary)) +
    geom_point()
```

As experience goes up, salary also goes up, as you would expect. Also, the trend seems more or less straight.

(c) Fit a regression predicting salary from experience, and display the results. Is the slope positive or negative? Does that make sense?

Solution

```
soc.1 = lm(salary ~ experience, data = soc)
summary(soc.1)
```

```
##
## Call:
## lm(formula = salary ~ experience, data = soc)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -17666.3  -5498.2   -726.7   4667.7  27811.6
##
## Coefficients:
##             Estimate Std. Error t value
## (Intercept)  11368.7     3160.3   3.597
## experience    2141.4      160.8  13.314
##             Pr(>|t|)
## (Intercept) 0.000758 ***
## experience   < 2e-16 ***
## ---
## Signif. codes:
##   0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1  ' ' 1
##
## Residual standard error: 8642 on 48 degrees of freedom
```

```
## Multiple R-squared:  0.7869, Adjusted R-squared:  0.7825
## F-statistic: 177.3 on 1 and 48 DF,  p-value: < 2.2e-16
```

The slope is (significantly) positive, which squares with our guess (more experience goes with greater salary), and also the upward trend on the scatterplot. The value of the slope is about 2,000; this means that one more year of experience goes with about a $2,000 increase in salary.

(d) Obtain and plot the residuals against the fitted values. What problem do you see?

Solution

The easiest way to do this with `ggplot` is to plot the *regression object* (even though it is not actually a data frame), and plot the `.fitted` and `.resid` columns in it, not forgetting the initial dots:

```
ggplot(soc.1, aes(x = .fitted, y = .resid)) +
    geom_point()
```

I see a "fanning-out": the residuals are getting bigger *in size* (further away from zero) as the fitted values get bigger. That is, when the (estimated) salary gets larger, it also gets more variable.

Fanning-out is sometimes hard to see. What you can do if you suspect that it might have happened is to plot the *absolute value* of the residuals against the fitted values. The absolute value is the residual without its plus or minus sign, so if the residuals are getting bigger in size, their absolute values are getting bigger. That would look like this:

```
ggplot(soc.1, aes(x = .fitted, y = abs(.resid))) +
    geom_point() + geom_smooth()
```

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

I added a smooth trend to this to help us judge whether the absolute-value-residuals are getting bigger as the fitted values get bigger. It looks to me as if the overall trend is an increasing one, apart from those few small fitted values that have larger-sized residuals. Don't get thrown off by the kinks in the smooth trend. Here is a smoother version:

```
ggplot(soc.1, aes(x = .fitted, y = abs(.resid))) +
    geom_point() + geom_smooth(span = 2)
```

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

The larger fitted values, according to this, have residuals larger in size.

The thing that controls the smoothness of the smooth trend is the value of `span` in `geom_smooth`. The default is 0.75. The larger the

value you use, the smoother the trend; the smaller, the more wiggly. I'm inclined to think that the default value is a bit too small. Possibly this value is too big, but it shows you the idea.

(e) The problem you unearthed in the previous part is often helped by a transformation. Run Box-Cox on your data to find a suitable transformation. What transformation is suggested?

Solution

You'll need to call in (and install if necessary) the package `MASS` that contains `boxcox`:

```
library(MASS)
```

```
##
## Attaching package: 'MASS'

## The following object is masked from 'package:dplyr':
##
##     select
```

I explain that "masked" thing below.

```
boxcox(salary ~ experience, data = soc)
```

That one looks like $\lambda = 0$ or log. You could probably also justify fourth root (power 0.25), but log is a very common transformation, which people won't need much persuasion to accept.

There's one annoyance with `MASS`: it has a `select` (which I have never used), and if you load `tidyverse` first and `MASS` second, as I have done here, when you mean to run the column-selection `select`, it will actually run the `select` that comes from `MASS`, and give you an error that you will have a terrible time debugging. That's what that "masked" message was when you loaded `MASS`.

So I'm going to be tidy and get rid of `MASS`, now that I'm finished with it. Let's first see which packages are loaded, rather a lot in my case:

The packages before *tidyverse* other than *MASS* are all loaded by the *tidyverse*, which is why there are so many.

```r
search()
```

```
##  [1] ".GlobalEnv"        "package:MASS"
##  [3] "package:broom"     "package:GGally"
##  [5] "package:bindrcpp"  "package:forcats"
##  [7] "package:stringr"   "package:dplyr"
##  [9] "package:purrr"     "package:readr"
## [11] "package:tidyr"     "package:tibble"
## [13] "package:ggplot2"   "package:tidyverse"
## [15] "package:stats"     "package:graphics"
## [17] "package:grDevices" "package:utils"
## [19] "package:datasets"  "package:methods"
## [21] "Autoloads"         "package:base"
```

then get rid of `MASS`:

```r
detach("package:MASS", unload = T)
```

Now check that it has gone:

```r
search()
```

```
##  [1] ".GlobalEnv"        "package:broom"
##  [3] "package:GGally"    "package:bindrcpp"
##  [5] "package:forcats"   "package:stringr"
##  [7] "package:dplyr"     "package:purrr"
##  [9] "package:readr"     "package:tidyr"
## [11] "package:tibble"    "package:ggplot2"
## [13] "package:tidyverse" "package:stats"
## [15] "package:graphics"  "package:grDevices"
## [17] "package:utils"     "package:datasets"
## [19] "package:methods"   "Autoloads"
## [21] "package:base"
```

It has. Now any calls to `select` will use the right one. We hope.

The output of `search` is called the **search list**, and it tells you where R will go looking for things. The first one `.GlobalEnv` is where all your variables, data frames etc. get stored, and that is what gets searched first. Then R will go looking in each thing in turn until it finds what it is looking for. When you load a package with `library()`, it gets added to the list *in second place*, behind `.GlobalEnv`. So, when we had `MASS` loaded (the first `search()`), if we called `select`, then it would find the one in `MASS` first.

All the ones that are part of this project, anyway.

That means that if you write a function with the same name as one that is built into R or a package, yours is the one that will get called. This is probably a bad idea, since you won't be able to get at R's function by that name.

If you want to insist on something like "the `select` that lives in `dplyr`", you can do that by saying `dplyr::select`. But this is kind of cumbersome if you don't need to do it, which is why I got rid of `MASS` here.

(f) Calculate a new variable as suggested by your transformation. Use your transformed response in a regression, showing the summary.

Solution

The best way is to add the new variable to the data frame using `mutate`, and save that new data frame. That goes like this:

```
soc.2 = soc %>% mutate(log_salary = log(salary))
```

and then

```
soc.3 = lm(log_salary ~ experience, data = soc.2)
summary(soc.3)
```

```
##
## Call:
## lm(formula = log_salary ~ experience, data = soc.2)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -0.35435 -0.09046 -0.01725  0.09739  0.26355
##
## Coefficients:
##             Estimate Std. Error t value
## (Intercept) 9.841315   0.056356  174.63
## experience  0.049979   0.002868   17.43
##             Pr(>|t|)
## (Intercept)   <2e-16 ***
## experience    <2e-16 ***
## ---
## Signif. codes:
##   0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1  ' ' 1
##
## Residual standard error: 0.1541 on 48 degrees of freedom
## Multiple R-squared:  0.8635, Adjusted R-squared:  0.8607
## F-statistic: 303.7 on 1 and 48 DF,  p-value: < 2.2e-16
```

I think it's best to save the data frame with `log_salary` in it, since we'll be doing a couple of things with it, and it's best to be able to start from `soc.2`. But you can also do this:

```
soc %>% mutate(log_salary = log(salary)) %>% lm(log_salary ~
    experience, data = .) %>% summary()
```

```
##
## Call:
```

```
## lm(formula = log_salary ~ experience, data = .)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -0.35435 -0.09046 -0.01725  0.09739  0.26355
##
## Coefficients:
##             Estimate Std. Error t value
## (Intercept) 9.841315   0.056356  174.63
## experience  0.049979   0.002868   17.43
##             Pr(>|t|)
## (Intercept)   <2e-16 ***
## experience    <2e-16 ***
## ---
## Signif. codes:
##   0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1  ' ' 1
##
## Residual standard error: 0.1541 on 48 degrees of freedom
## Multiple R-squared:  0.8635, Adjusted R-squared:  0.8607
## F-statistic: 303.7 on 1 and 48 DF,  p-value: < 2.2e-16
```

The second line is where the fun starts: `lm` wants the data frame as a `data=` at the end. So, to specify a data frame in something like `lm`, we have to use the special symbol `.`, which is another way to say "the data frame that came out of the previous step".

Got that? All right. The last line is a piece of cake in comparison. Normally `summary` would require a data frame or a fitted model object, but the second line produces one (a fitted model object) as output, which goes into `summary` as the first (and only) thing, so all is good and we get the regression output.

What we lose by doing this is that if we need something later from this fitted model object, we are out of luck since we didn't save it. That's why I created `soc.2` and `soc.3` above.

You can also put functions of things directly into `lm`:

```
soc.1a = lm(log(salary) ~ experience, data = soc)
summary(soc.1a)
```

```
##
## Call:
## lm(formula = log(salary) ~ experience, data = soc)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -0.35435 -0.09046 -0.01725  0.09739  0.26355
```

```
##
## Coefficients:
##             Estimate Std. Error t value
## (Intercept) 9.841315   0.056356  174.63
## experience  0.049979   0.002868   17.43
##             Pr(>|t|)
## (Intercept)   <2e-16 ***
## experience    <2e-16 ***
## ---
## Signif. codes:
##   0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1  ' ' 1
##
## Residual standard error: 0.1541 on 48 degrees of freedom
## Multiple R-squared:  0.8635, Adjusted R-squared:  0.8607
## F-statistic: 303.7 on 1 and 48 DF,  p-value: < 2.2e-16
```

(g) Obtain and plot the residuals against the fitted values for this regression. Do you seem to have solved the problem with the previous residual plot?

Solution

As we did before, treating the regression object as if it were a data frame:

```
ggplot(soc.3, aes(x = .fitted, y = .resid)) +
    geom_point()
```

That, to my mind, is a horizontal band of points, so I would say yes, I have solved the fanning out.

One concern I have about the residuals is that there seem to be a couple of very negative values: that is, are the residuals normally distributed as they should be? Well, that's easy enough to check:

```
ggplot(soc.3, aes(sample = .resid)) + stat_qq() +
    stat_qq_line()
```

The issues here are that those bottom two values are a bit too low, and the top few values are a bit bunched up (that curve at the top). It is really not bad, though, so I am making the call that I don't think I needed to worry. Note that the transformation we found here is the same as the log-salary used by the management consultants in the backward-elimination question, and with the same effect: an extra year of experience goes with a *percent* increase in salary.

What increase? Well, the slope is about 0.05, so adding a year of experience is predicted to increase log-salary by 0.05, or to multiply actual salary by

```
exp(0.05)
```

```
## [1] 1.051271
```

or to increase salary by about 5%.

Mathematically, e^x is approximately $1 + x$ for small x, which winds up meaning that the slope in a model like this, if it is small, indicates about the percent increase in the response associated with a 1-unit change in the explanatory variable. Note that this only works with e^x and natural logs, not base 10 logs or anything like that.

13.8 *Predicting volume of wood in pine trees*

In forestry, the financial value of a tree is the volume of wood that it contains. This is difficult to estimate while the tree is still standing, but the diameter is easy to measure with a tape measure (to measure the circumference) and a calculation involving π, assuming that the cross-section of the tree is at least approximately circular. The standard measurement is "diameter at breast height" (that is, at the height of a human breast or chest), defined as being 4.5 feet above the ground.

Several pine trees had their diameter measured shortly before being cut down, and for each tree, the volume of wood was recorded.

The data are in link. The diameter is in inches and the volume is in cubic inches. Is it possible to predict the volume of wood from the diameter?

(a) Read the data into R and display the values (there are not very many).

Solution

Observe that the data values are separated by spaces, and therefore that `read_delim` will do it:

```
my_url = "http://www.utsc.utoronto.ca/~butler/c32/pinetrees.txt"
trees = read_delim(my_url, " ")
```

```
## Parsed with column specification:
## cols(
##   diameter = col_integer(),
##   volume = col_integer()
## )
```

```
trees
```

```
## # A tibble: 10 x 2
##    diameter volume
##       <int>  <int>
##  1       32    185
##  2       29    109
##  3       24     95
##  4       45    300
##  5       20     30
##  6       30    125
##  7       26     55
##  8       40    246
##  9       24     60
## 10       18     15
```

That looks like the data file.

(b) Make a suitable plot.

Solution

No clues this time. You need to recognize that you have two quantitative variables, so that a scatterplot is called for. Also, the volume is the response, so that should go on the y-axis:

```
ggplot(trees, aes(x = diameter, y = volume)) +
    geom_point()
```

You can put a smooth trend on it if you like, which would look like this:

```
ggplot(trees, aes(x = diameter, y = volume)) +
    geom_point() + geom_smooth()
```

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

I'll take either of those for this part, though I think the smooth trend actually obscures the issue here (because there is not so much data).

(c) Describe what you learn from your plot about the relationship between diameter and volume, if anything.

Solution

The word "relationship" offers a clue that a scatterplot would have been a good idea, if you hadn't realized by now. I am guided by "form, direction, strength" in looking at a scatterplot:

- Form: it is an apparently linear relationship.
- Direction: it is an upward trend: that is, a tree with a larger diameter also has a larger volume of wood. (This is not very surprising.)
- Strength: I'd call this a strong (or moderate-to-strong) relationship. (We'll see in a minute what the R-squared is.)

You don't need to be as formal as this, but you *do* need to get at the idea that it is an upward trend, apparently linear, and at least fairly strong.

When this was graded, it was 3 marks, to clue you in that there are three things to say.

(d) Fit a (linear) regression, predicting volume from diameter, and obtain the `summary`. How would you describe the R-squared?

Solution

My naming convention is (usually) to call the fitted model object by the name of the response variable and a number. (I have always used dots, but in the spirit of the `tidyverse` I suppose I should use underscores.)

```
volume.1 = lm(volume ~ diameter, data = trees)
summary(volume.1)
```

```
##
## Call:
## lm(formula = volume ~ diameter, data = trees)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -36.497  -9.982   1.751   8.959  28.139
##
## Coefficients:
##             Estimate Std. Error t value
## (Intercept) -191.749     23.954  -8.005
## diameter      10.894      0.801  13.600
##             Pr(>|t|)
## (Intercept) 4.35e-05 ***
## diameter    8.22e-07 ***
## ---
## Signif. codes:
##   0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1  ' ' 1
##
## Residual standard error: 20.38 on 8 degrees of freedom
## Multiple R-squared:  0.9585, Adjusted R-squared:  0.9534
## F-statistic:   185 on 1 and 8 DF,  p-value: 8.217e-07
```

R-squared is nearly 96%, so the relationship is definitely a strong one.

I also wanted to mention the `broom` package, which was installed with the `tidyverse` but which you need to load separately. It provides two handy ways to summarize a fitted model (regression, analysis of variance or whatever):

```
library(broom)
glance(volume.1)
```

```
## # A tibble: 1 x 11
##   r.squared adj.r.squared sigma statistic
## *     <dbl>         <dbl> <dbl>     <dbl>
## 1     0.959         0.953  20.4      185.
## # ... with 7 more variables: p.value <dbl>,
## #   df <int>, logLik <dbl>, AIC <dbl>,
## #   BIC <dbl>, deviance <dbl>,
## #   df.residual <int>
```

This gives a one-line summary of a model, including things like R-squared. This is handy if you're fitting more than one model, because you can collect the one-line summaries together into a data frame and eyeball them.

The other summary is this one:

```
tidy(volume.1)
```

```
## # A tibble: 2 x 5
##   term  estimate std.error statistic p.value
##   <chr>    <dbl>     <dbl>     <dbl>   <dbl>
## 1 (Int~   -192.     24.0       -8.01 4.35e-5
## 2 diam~     10.9     0.801     13.6  8.22e-7
```

This gives a table of intercepts, slopes and their P-values, but the value to this one is that it is a *data frame*, so if you want to pull anything out of it, you know how to do that:

The *summary* output is more designed for looking at than for extracting things from.

```
tidy(volume.1) %>% filter(term == "diameter")
```

```
## # A tibble: 1 x 5
##   term  estimate std.error statistic p.value
##   <chr>    <dbl>     <dbl>     <dbl>   <dbl>
## 1 diam~     10.9     0.801      13.6 8.22e-7
```

This gets the estimated slope and its P-value, without worrying about the corresponding things for the intercept, which are usually of less interest anyway.

(e) Draw a graph that will help you decide whether you trust the linearity of this regression. What do you conclude? Explain briefly.

Solution

The thing I'm fishing for is a residual plot (of the residuals against the fitted values), and on it you are looking for a random mess of nothingness:

```
ggplot(volume.1, aes(x = .fitted, y = .resid)) +
    geom_point()
```

Make a call. You could say that there's no discernible pattern, especially with such a small data set, and therefore that the regression is fine. Or you could say that there is fanning-in: the two points on the right have residuals close to 0 while the points on the left have residuals larger in size. Say something.

I don't think you can justify a curve or a trend, because the residuals on the left are both positive and negative.

My feeling is that the residuals on the right are close to 0 because these points have noticeably larger diameter than the others, and they are *influential* points in the regression that will pull the line closer to themselves. This is why their residuals are close to zero. But I am happy with either of the points made in the paragraph under the plot.

(f) What would you guess would be the volume of a tree of diameter zero? Is that what the regression predicts? Explain briefly.

Solution

Logically, a tree that has diameter zero is a non-existent tree, so its volume should be zero as well. In the regression, the quantity that says what volume is when diameter is zero is the *intercept*. Here the

intercept is -192, which is definitely not zero. In fact, if you look at the P-value, the intercept is significantly *less* than zero. Thus, the model makes no logical sense for trees of small diameter. The smallest tree in the data set has diameter 18, which is not really small, I suppose, but it is a little disconcerting to have a model that makes no logical sense.

(g) A simple way of modelling a tree's shape is to pretend it is a cone, like this, but probably taller and skinnier:

with its base on the ground. What is the relationship between the *diameter* (at the base) and volume of a cone? (If you don't remember, look it up. You'll probably get a formula in terms of the radius, which you'll have to convert. Cite the website you used.)

Solution

According to link, the volume of a cone is $V = \pi r^2 h/3$, where V is the volume, r is the radius (at the bottom of the cone) and h is the height. The diameter is twice the radius, so replace r by $d/2$, d being the diameter. A little algebra gives

$$V = \pi d^2 h/12.$$

(h) Fit a regression model that predicts volume from diameter according to the formula you obtained in the previous part. You can

assume that the trees in this data set are of similar heights, so that the height can be treated as a constant.

Display the results.

Solution

According to my formula, the volume depends on the diameter squared, which I include in the model thus:

```
volume.2 = lm(volume ~ I(diameter^2), data = trees)
summary(volume.2)
```

```
##
## Call:
## lm(formula = volume ~ I(diameter^2), data = trees)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -29.708  -9.065  -5.722   3.032  40.816
##
## Coefficients:
##                Estimate Std. Error t value
## (Intercept)   -30.82634   13.82243   -2.23
## I(diameter^2)   0.17091    0.01342   12.74
##               Pr(>|t|)
## (Intercept)     0.0563 .
## I(diameter^2) 1.36e-06 ***
## ---
## Signif. codes:
##   0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1  ' ' 1
##
## Residual standard error: 21.7 on 8 degrees of freedom
## Multiple R-squared:  0.953,  Adjusted R-squared:  0.9471
## F-statistic: 162.2 on 1 and 8 DF,  p-value: 1.359e-06
```

This adds an intercept as well, which is fine (there are technical difficulties around removing the intercept).

That's as far as I wanted you to go, but (of course) I have a few comments.

The intercept here is still negative, but not significantly different from zero, which is a step forward. The R-squared for this regression is very similar to that from our linear model (the one for which the intercept made no sense). So, from that point of view, either model predicts the data well. I should look at the residuals from this one:

```
ggplot(volume.2, aes(x = .fitted, y = .resid)) +
    geom_point()
```

I really don't think there are any problems there.

Now, I said to assume that the trees are all of similar height. This seems entirely questionable, since the trees vary quite a bit in diameter, and you would guess that trees with bigger diameter would also be taller. It seems more plausible that the same kind of trees (pine trees in this case) would have the same "shape", so that if you knew the diameter you could *predict* the height, with larger-diameter trees being taller. Except that we don't have the heights here, so we can't build a model for that.

So I went looking in the literature. I found this paper: link. This gives several models for relationships between volume, diameter and height. In the formulas below, there is an implied "plus error" on the right, and the α_i are parameters to be estimated.

For predicting height from diameter (equation 1 in paper):

$$h = \exp(\alpha_1 + \alpha_2 d^{\alpha_3})$$

For predicting volume from height and diameter (equation 6):

$$V = \alpha_1 d^{\alpha_2} h^{\alpha_3}$$

This is a take-off on our assumption that the trees were cone-shaped, with cone-shaped trees having $\alpha_1 = \pi/12$, $\alpha_2 = 2$ and $\alpha_3 = 1$. The paper uses different units, so α_1 is not comparable, but α_2 and α_3 are (as estimated from the data in the paper, which were for longleaf pine) quite close to 2 and 1.

Last, the actual relationship that helps us: predicting volume from just diameter (equation 5):

$$V = \alpha_1 d^{\alpha_2}$$

This is a power law type of relationship. For example, if you were

willing to pretend that a tree was a cone with height proportional to diameter (one way of getting at the idea of a bigger tree typically being taller, instead of assuming constant height as we did), that would imply $\alpha_2 = 3$ here.

This is non-linear as it stands, but we can bash it into shape by taking logs:

$$\ln V = \ln \alpha_1 + \alpha_2 \ln d$$

so that log-volume has a linear relationship with log-diameter and we can go ahead and estimate it:

```
volume.3 = lm(log(volume) ~ log(diameter), data = trees)
summary(volume.3)
```

```
##
## Call:
## lm(formula = log(volume) ~ log(diameter), data = trees)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -0.40989 -0.22341  0.01504  0.10459  0.53596
##
## Coefficients:
##               Estimate Std. Error t value
## (Intercept)    -5.9243     1.1759  -5.038
## log(diameter)   3.1284     0.3527   8.870
##               Pr(>|t|)
## (Intercept)      0.001 **
## log(diameter) 2.06e-05 ***
## ---
## Signif. codes:
##   0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1  ' ' 1
##
## Residual standard error: 0.3027 on 8 degrees of freedom
## Multiple R-squared:  0.9077, Adjusted R-squared:  0.8962
## F-statistic: 78.68 on 1 and 8 DF,  p-value: 2.061e-05
```

The parameter that I called α_2 above is the slope of this model, 3.13. This is a bit different from the figure in the paper, which was 2.19. I think these are comparable even though the other parameter is not (again, measurements in different units, plus, this time we need to take the log of it). I think the "slopes" are comparable because we haven't estimated our slope all that accurately:

```
confint(volume.3)
```

```
##                    2.5 %    97.5 %
## (Intercept)   -8.635791 -3.212752
## log(diameter)  2.315115  3.941665
```

From 2.3 to 3.9. It is definitely not zero, but we are rather less sure about what it is, and 2.19 is not completely implausible.

The R-squared here, though it is less than the other ones we got, is still high. The residuals are these:

```
ggplot(volume.3, aes(x = .fitted, y = .resid)) +
    geom_point()
```

which again seem to show no problems. The residuals are smaller in size now because of the log transformation: the actual and predicted log-volumes are smaller numbers than the actual and predicted volumes, so the residuals are now closer to zero.

Does this model behave itself at zero? Well, roughly at least: if the diameter is very small, its log is very negative, and the predicted log-volume is also very negative (the slope is positive). So the predicted actual volume will be close to zero. If you want to make that mathematically rigorous, you can take limits, but that's the intuition. We can also do some predictions: set up a data frame that has a column called `diameter` with some diameters to predict for:

```
d = tibble(diameter = c(1, 2, seq(5, 50, 5)))
d
```

```
## # A tibble: 12 x 1
##    diameter
##       <dbl>
##  1        1
##  2        2
```

```
## 3        5
## 4       10
## 5       15
## 6       20
## 7       25
## 8       30
## 9       35
## 10      40
## 11      45
## 12      50
```

and then feed that into `predict`:

```
p = predict(volume.3, d)
d %>% mutate(pred = p)
```

```
## # A tibble: 12 x 2
##    diameter   pred
##       <dbl>  <dbl>
## 1        1 -5.92
## 2        2 -3.76
## 3        5 -0.889
## 4       10  1.28
## 5       15  2.55
## 6       20  3.45
## 7       25  4.15
## 8       30  4.72
## 9       35  5.20
## 10      40  5.62
## 11      45  5.98
## 12      50  6.31
```

These are predicted log-volumes, so we'd better anti-log them. `log` in R is natural logs, so this is inverted using `exp`:

```
d %>% mutate(pred = exp(p))
```

```
## # A tibble: 12 x 2
##    diameter      pred
##       <dbl>     <dbl>
## 1        1   0.00267
## 2        2   0.0234
## 3        5   0.411
## 4       10   3.59
## 5       15  12.8
## 6       20  31.4
```

```
##  7       25  63.2
##  8       30 112.
##  9       35 181.
## 10       40 275.
## 11       45 397.
## 12       50 552.
```

For a diameter near zero, the predicted volume appears to be near zero as well.

I mentioned `broom` earlier. We can make a data frame out of the one-line summaries of our three models:

```
bind_rows(glance(volume.1), glance(volume.2),
    glance(volume.3))
```

```
## # A tibble: 3 x 11
##   r.squared adj.r.squared  sigma statistic
##       <dbl>         <dbl>  <dbl>     <dbl>
## 1     0.959         0.953 20.4       185.
## 2     0.953         0.947 21.7       162.
## 3     0.908         0.896  0.303      78.7
## # ... with 7 more variables: p.value <dbl>,
## #   df <int>, logLik <dbl>, AIC <dbl>,
## #   BIC <dbl>, deviance <dbl>,
## #   df.residual <int>
```

(I mistakenly put `glimpse` instead of `glance` there the first time. The former is for a quick look at a *data frame*, while the latter is for a quick look at a *model*.)

The three R-squareds are all high, with the one from the third model being a bit lower as we saw before.

My code is rather repetitious. There has to be a way to streamline it. I was determined to find out how. My solution involves putting the three models in a `list`, and then using `map` to get the `glance` output for each one, and `bind_rows` to glue the results together into one data frame. I was inspired to try this by remembering that `map_df` will work for a function like `glance` that outputs a data frame:

```
model_list = list(volume.1, volume.2, volume.3)
map_df(model_list, ~glance(.))
```

```
## # A tibble: 3 x 11
##   r.squared adj.r.squared  sigma statistic
##       <dbl>         <dbl>  <dbl>     <dbl>
## 1     0.959         0.953 20.4       185.
## 2     0.953         0.947 21.7       162.
```

```
## 3     0.908         0.896  0.303      78.7
## # ... with 7 more variables: p.value <dbl>,
## #   df <int>, logLik <dbl>, AIC <dbl>,
## #   BIC <dbl>, deviance <dbl>,
## #   df.residual <int>
```

It works. You see the three R-squared values in the first column. The third model is otherwise a lot different from the others because it has a different response variable.

Other thoughts:

How might you measure or estimate the height of a tree (other than by climbing it and dropping a tape measure down)? One way, that works if the tree is fairly isolated, is to walk away from its base. Periodically, you point at the top of the tree, and when the angle between your arm and the ground reaches 45 degrees, you stop walking. (If it's greater than 45 degrees, you walk further away, and if it's less, you walk back towards the tree.) The distance between you and the base of the tree is then equal to the height of the tree, and if you have a long enough tape measure you can measure it.

The above works because the tangent of 45 degrees is 1. If you have a device that will measure the actual angle, you can be any distance away from the tree, point the device at the top, record the angle, and do some trigonometry to estimate the height of the tree (to which you add the height of your eyes).

These days, there are apps that will let you do this with your phone. I found one called Clinometer. See also [link](https://gabrielhemery.com/how-to-calculate-tree-height-using-a-smartphone/).

13.9 *Tortoise shells and eggs*

A biologist measured the length of the carapace (shell) of female tortoises, and then x-rayed the tortoises to count how many eggs they were carrying. The length is measured in millimetres. The data are in link. The biologist is wondering what kind of relationship, if any, there is between the carapace length (as an explanatory variable) and the number of eggs (as a response variable).

(a) Read in the data, and check that your values look reasonable.

Solution

Look at the data first. The columns are aligned and separated by more than one space, so it's `read_table`:

```
my_url = "http://www.utsc.utoronto.ca/~butler/d29/tortoise-eggs.txt"
tortoises = read_table(my_url)
```

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

```
##   eggs = col_integer()
## )
```

```
tortoises
```

```
## # A tibble: 18 x 2
##    length  eggs
##     <int> <int>
##  1    284     3
##  2    290     2
##  3    290     7
##  4    290     7
##  5    298    11
##  6    299    12
##  7    302    10
##  8    306     8
##  9    306     8
## 10    309     9
## 11    310    10
## 12    311    13
## 13    317     7
## 14    317     9
## 15    320     6
## 16    323    13
## 17    334     2
## 18    334     8
```

Those look the same as the values in the data file. (*Some* comment is needed here. I don't much mind what, but something that suggests that you have eyeballed the data and there are no obvious problems: that is what I am looking for.)

(b) Obtain a scatterplot, with a smooth trend, of the data.

Solution
Something like this:

```
ggplot(tortoises, aes(x = length, y = eggs)) +
    geom_point() + geom_smooth()
```

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

(c) The biologist expected that a larger tortoise would be able to carry more eggs. Is that what the scatterplot is suggesting? Explain briefly why or why not.

Solution

The biologist's expectation is of an upward trend. But it looks as if the trend on the scatterplot is up, then down, ie. a curve rather than a straight line. So this is not what the biologist was expecting.

(d) Fit a straight-line relationship and display the summary.

Solution

```
tortoises.1 = lm(eggs ~ length, data = tortoises)
summary(tortoises.1)
```

```
##
## Call:
## lm(formula = eggs ~ length, data = tortoises)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -6.7790 -1.1772 -0.0065  2.0487  4.8556
##
## Coefficients:
##             Estimate Std. Error t value
## (Intercept) -0.43532   17.34992  -0.025
## length       0.02759    0.05631   0.490
##             Pr(>|t|)
## (Intercept)    0.980
## length         0.631
```

```
##
## Residual standard error: 3.411 on 16 degrees of freedom
## Multiple R-squared:  0.01478,    Adjusted R-squared:  -0.0468
## F-statistic:  0.24 on 1 and 16 DF,  p-value: 0.6308
```

I didn't ask for a comment, but feel free to observe that this regression is truly awful, with an R-squared of less than 2% and a non-significant effect of `length`.

(e) Add a squared term to your regression, fit that and display the summary.

Solution

The `I()` is needed because the raise-to-a-power symbol has a special meaning in a model formula, and we want to *not* use that special meaning:

```
tortoises.2 = lm(eggs ~ length + I(length^2),
    data = tortoises)
summary(tortoises.2)
```

```
##
## Call:
## lm(formula = eggs ~ length + I(length^2), data = tortoises)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -4.0091 -1.8480 -0.1896  2.0989  4.3605
##
## Coefficients:
##               Estimate Std. Error t value
## (Intercept) -8.999e+02  2.703e+02  -3.329
## length       5.857e+00  1.750e+00   3.347
## I(length^2) -9.425e-03  2.829e-03  -3.332
##             Pr(>|t|)
## (Intercept)  0.00457 **
## length       0.00441 **
## I(length^2)  0.00455 **
## ---
## Signif. codes:
##   0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1  ' ' 1
##
## Residual standard error: 2.671 on 15 degrees of freedom
## Multiple R-squared:  0.4338, Adjusted R-squared:  0.3583
## F-statistic: 5.747 on 2 and 15 DF,  p-value: 0.01403
```

Another way is to use `update`:

```r
tortoises.2a = update(tortoises.1, . ~ . + I(length^2))
summary(tortoises.2a)
```

```
##
## Call:
## lm(formula = eggs ~ length + I(length^2), data = tortoises)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -4.0091 -1.8480 -0.1896  2.0989  4.3605
##
## Coefficients:
##               Estimate Std. Error t value
## (Intercept) -8.999e+02  2.703e+02  -3.329
## length       5.857e+00  1.750e+00   3.347
## I(length^2) -9.425e-03  2.829e-03  -3.332
##             Pr(>|t|)
## (Intercept)  0.00457 **
## length       0.00441 **
## I(length^2)  0.00455 **
## ---
## Signif. codes:
##   0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1  ' ' 1
##
## Residual standard error: 2.671 on 15 degrees of freedom
## Multiple R-squared:  0.4338, Adjusted R-squared:  0.3583
## F-statistic: 5.747 on 2 and 15 DF,  p-value: 0.01403
```

(f) Is a curve better than a line for these data? Justify your answer in two ways: by comparing a measure of fit, and by doing a suitable test of significance.

Solution

An appropriate measure of fit is R-squared. For the straight line, this is about 0.01, and for the regression with the squared term it is about 0.43. This tells us that a straight line fits appallingly badly, and that a curve fits a *lot* better. This doesn't do a test, though. For that, look at the slope of the length-squared term in the second regression; in particular, look at its P-value. This is 0.0045, which is small: the squared term is necessary, and taking it out would be a mistake. The relationship really is curved, and trying to describe it with a straight line would be a big mistake.

(g) Make a residual plot for the straight line model: that is, plot the residuals against the fitted values. Does this echo your conclusions of the previous part? In what way? Explain briefly.

Solution

Plot the things called `.fitted` and `.resid` from the regression object, which is not a data frame but you can treat it as if it is for this:

```
ggplot(tortoises.1, aes(x = .fitted, y = .resid)) +
    geom_point()
```

Up to you whether you put a smooth trend on it or not:

```
ggplot(tortoises.1, aes(x = .fitted, y = .resid)) +
    geom_point() + geom_smooth()
```

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

Looking at the plot, you see a curve, up and down. The most negative residuals go with small or large fitted values; when the fitted value is in the middle, the residual is usually positive. A curve on the residual plot indicates a curve in the actual relationship. We just found above that a curve does fit a lot better, so this is all consistent.

Aside: the grey "envelope" is wide, so there is a lot of scatter on the residual plot. The grey envelope almost contains zero all the way across, so the evidence for a curve (or any other kind of trend) is not all that strong, based on this plot. This is in great contrast to the regression with length-squared, where the length-squared term is *definitely* necessary.

That was all I wanted, but you can certainly look at other plots. Normal quantile plot of the residuals:

```
ggplot(tortoises.1, aes(sample = .resid)) + stat_qq() +
    stat_qq_line()
```

This is not the best: the low values are a bit too low, so that the whole picture is (a little) skewed to the left.

Another plot you can make is to assess fan-out: you plot the *absolute value* of the residuals against the fitted values. The idea is that if there is fan-out, the absolute value of the residuals will get bigger:

The very negative residuals are at the left and right of the residual plot; they are there because the relationship is a curve. If you were to look at the residuals from the model with length-squared, you probably wouldn't see this.

The value, but throw away the minus sign if it has one.

```
ggplot(tortoises.1, aes(x = .fitted, y = abs(.resid))) +
    geom_point() + geom_smooth()
```

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

I put the smooth curve on as a kind of warning: it looks as if the size of the residuals goes down and then up again as the fitted values increase. But the width of the grey "envelope" and the general scatter of the points suggests that there is really not much happening here at all. On a plot of residuals, the grey envelope is really more informative than the blue smooth trend. On this one, there is no evidence of any fan-out (or fan-in).

13.10 *Crickets revisited*

This is a continuation of the crickets problem that you may have seen before (minus the data tidying).

Male tree crickets produce "mating songs" by rubbing their wings together to produce a chirping sound. It is hypothesized that female tree crickets identify males of the correct species by how fast (in chirps per second) the male's mating song is. This is called the "pulse rate". Some data for two species of crickets are in link as a CSV file. The columns are species (text), temperature, and pulse rate (numbers). This is the tidied version of the data set that the previous version of this question had you create. The research question is whether males of the different species have different average pulse rates. It is also of interest to see whether temperature has an effect, and if so, what.

(a) Read the data into R and display what you have.

Solution

Nothing terribly surprising here:

```
my_url = "http://www.utsc.utoronto.ca/~butler/c32/crickets2.csv"
crickets = read_csv(my_url)
```

```
## Parsed with column specification:
## cols(
##   species = col_character(),
##   temperature = col_double(),
##   pulse_rate = col_double()
## )
```

```
crickets
```

```
## # A tibble: 31 x 3
##    species       temperature pulse_rate
##    <chr>               <dbl>      <dbl>
##  1 exclamationis        20.8       67.9
##  2 exclamationis        20.8       65.1
##  3 exclamationis        24         77.3
##  4 exclamationis        24         78.7
##  5 exclamationis        24         79.4
##  6 exclamationis        24         80.4
##  7 exclamationis        26.2       85.8
##  8 exclamationis        26.2       86.6
##  9 exclamationis        26.2       87.5
## 10 exclamationis        26.2       89.1
## # ... with 21 more rows
```

31 crickets, which is what I remember. What species are there?

```
crickets %>% count(species)
```

```
## # A tibble: 2 x 2
##   species           n
##   <chr>         <int>
## 1 exclamationis    14
## 2 niveus           17
```

That looks good. We proceed.

(b) Do a two-sample t-test to see whether the mean pulse rates differ between species. What do you conclude?

Solution

Drag your mind way back to this:

```
t.test(pulse_rate ~ species, data = crickets)
```

```
##
##  Welch Two Sample t-test
##
```

```
## data:  pulse_rate by species
## t = 5.2236, df = 28.719, p-value =
## 1.401e-05
## alternative hypothesis: true difference in means is not equal to 0
## 95 percent confidence interval:
##  14.08583 32.22677
## sample estimates:
## mean in group exclamationis
##                    85.58571
##        mean in group niveus
##                    62.42941
```

There is strong evidence of a difference in means (a P-value around 0.00001), and the confidence interval says that the mean chirp rate is higher for *exclamationis*. That is, not just for the crickets that were observed here, but for *all* crickets of these two species.

(c) Can you do that two-sample t-test as a regression?

Solution

Hang onto the "pulse rate depends on species" idea and try that in `lm`:

```
pulse.0 = lm(pulse_rate ~ species, data = crickets)
summary(pulse.0)
```

```
##
## Call:
## lm(formula = pulse_rate ~ species, data = crickets)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -20.486  -9.458  -1.729  13.342  22.271
##
## Coefficients:
##               Estimate Std. Error t value
## (Intercept)     85.586      3.316  25.807
## speciesniveus  -23.156      4.478  -5.171
##               Pr(>|t|)
## (Intercept)    < 2e-16 ***
## speciesniveus 1.58e-05 ***
## ---
## Signif. codes:
##   0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1  ' ' 1
##
## Residual standard error: 12.41 on 29 degrees of freedom
```

```
## Multiple R-squared:  0.4797, Adjusted R-squared:  0.4617
## F-statistic: 26.74 on 1 and 29 DF,  p-value: 1.579e-05
```

I had to use "model 0" for this since I already have a `pulse.1` below and I didn't want to go down and renumber everything.

Look along the `speciesniveus` line. Ignoring the fact that it is negative, the t-statistic is almost the same as before (5.17 vs. 5.22) and so is the P-value (1.4×10^{-5} vs. 1.6×10^{-5}).

Why aren't they exactly the same? Regression is assuming equal variances everywhere (that is, within the two `species`), and before, we did the Welch-Satterthwaite test that does not assume equal variances. What if we do the pooled t-test instead?

```r
t.test(pulse_rate ~ species, data = crickets,
    var.equal = T)
```

```
##
##  Two Sample t-test
##
## data:  pulse_rate by species
## t = 5.1706, df = 29, p-value =
## 1.579e-05
## alternative hypothesis: true difference in means is not equal to 0
## 95 percent confidence interval:
##  13.99690 32.31571
## sample estimates:
## mean in group exclamationis
##                    85.58571
##        mean in group niveus
##                    62.42941
```

Now the regression and the t-test *do* give exactly the same answers. We'll think about that equal-spreads assumption again later.

(d) The analysis in the last part did not use temperature, however. Is it possible that temperature also has an effect? To assess this, draw a scatterplot of pulse rate against temperature, with the points distinguished, somehow, by the species they are from.

This was the actual reason I thought of this question originally: I wanted you to do this.

Solution

One of the wonderful things about `ggplot` is that doing the obvious thing works:

```r
ggplot(crickets, aes(x = temperature, y = pulse_rate,
    colour = species)) + geom_point()
```

(e) What does the plot tell you that the t-test doesn't? How would you describe differences in pulse rates between species now?

Solution

The plot tells you that (for both species) as temperature goes up, pulse rate goes up as well. *Allowing for that*, the difference in pulse rates between the two species is even clearer than it was before. To see an example, pick a temperature, and note that the mean pulse rate at that temperature seems to be at least 10 higher for *exclamationis*, with a high degree of consistency. The t-test mixed up all the pulse rates at all the different temperatures. Even though the conclusion was clear enough, it could be clearer if we incorporated temperature into the analysis. There was also a potential source of unfairness in that the *exclamationis* crickets tended to be observed at higher temperatures than *niveus* crickets; since pulse rates increase with temperature, the apparent difference in pulse rates between the species might have been explainable by one species being observed mainly in higher temperatures. This was *utterly invisible* to us when we did the t-test, but it shows the importance of accounting for all the relevant variables when you do your analysis. If the species had been observed at opposite temperatures, we might have concluded that *niveus* have the higher pulse rates on average. I come back to this later when I discuss the confidence interval for species difference that comes out of the regression model with temperature.

And it shows the value of looking at relevant plots.

Mistakenly.

(f) Fit a regression predicting pulse rate from species and temperature. Compare the P-value for species in this regression to the one from the t-test. What does that tell you?

Solution

This is actually a so-called "analysis of covariance model", which properly belongs in D29, but it's really just a regression:

```
pulse.1 = lm(pulse_rate ~ species + temperature,
    data = crickets)
summary(pulse.1)
```

```
##
## Call:
## lm(formula = pulse_rate ~ species + temperature, data = crickets)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -3.0128 -1.1296 -0.3912  0.9650  3.7800
##
## Coefficients:
##                Estimate Std. Error t value
## (Intercept)    -7.21091    2.55094  -2.827
## speciesniveus -10.06529    0.73526 -13.689
## temperature     3.60275    0.09729  37.032
##               Pr(>|t|)
## (Intercept)    0.00858 **
## speciesniveus 6.27e-14 ***
## temperature    < 2e-16 ***
## ---
## Signif. codes:
##   0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1  ' ' 1
##
## Residual standard error: 1.786 on 28 degrees of freedom
## Multiple R-squared:  0.9896, Adjusted R-squared:  0.9888
## F-statistic:  1331 on 2 and 28 DF,  p-value: < 2.2e-16
```

The P-value for species is now 6.27×10^{-14} or 0.00000000000006, which is even less than the P-value of 0.00001 that came out of the t-test. That is to say, when you know temperature, you can be even more sure of your conclusion that there is a difference between the species.

The R-squared for this regression is almost 99%, which says that if you know both temperature and species, you can predict the pulse rate almost exactly.

In the regression output, the slope for species is about -10. It is labelled `speciesniveus`. Since species is categorical, `lm` uses the first category, *exclamationis*, as the baseline and expresses each other species relative to that. Since the slope is about -10, it says that at any given temperature, the mean pulse rate for *niveus* is about 10 less

than for *exclamationis*. This is pretty much what the scatterplot told us.

We can go a little further here:

```
confint(pulse.1)
```

```
##                    2.5 %    97.5 %
## (Intercept)   -12.436265 -1.985547
## speciesniveus -11.571408 -8.559175
## temperature     3.403467  3.802038
```

The second line says that the pulse rate for *niveus* is between about 8.5 and 11.5 less than for *exclamationis*, at any given temperature (comparing the two species at the same temperature as each other, but that temperature could be anything). This is a lot shorter than the CI that came out of the *t*-test, that went from 14 to 32. This is because we are now accounting for temperature, which also makes a difference. (In the *t*-test, the temperatures were all mixed up). What we also see is that the *t*-interval is shifted up compared to the one from the regression. This is because the *t*-interval conflates two things: the *exclamationis* crickets do have a higher pulse rate, but they were also observed at higher temperatures, which makes it look as if their pulse rates are more higher than they really are, when you account for temperature.

Mixes up.

This is actually grammatically correct.

This particular model constrains the slope with temperature to be the same for both species (just the intercepts differ). If you want to allow the slopes to differ between species, you add an interaction between temperature and species:

```
pulse.2 = lm(pulse_rate ~ species * temperature,
    data = crickets)
summary(pulse.2)
```

```
##
## Call:
## lm(formula = pulse_rate ~ species * temperature, data = crickets)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -3.7031 -1.3417 -0.1235  0.8100  3.6330
##
## Coefficients:
##                           Estimate
## (Intercept)               -11.0408
## speciesniveus              -4.3484
## temperature                 3.7514
```

```
## speciesniveus:temperature  -0.2340
##                            Std. Error t value
## (Intercept)                    4.1515  -2.659
## speciesniveus                  4.9617  -0.876
## temperature                    0.1601  23.429
## speciesniveus:temperature      0.2009  -1.165
##                            Pr(>|t|)
## (Intercept)                   0.013 *
## speciesniveus                 0.389
## temperature                  <2e-16 ***
## speciesniveus:temperature     0.254
## ---
## Signif. codes:
##   0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1  ' ' 1
##
## Residual standard error: 1.775 on 27 degrees of freedom
## Multiple R-squared:  0.9901, Adjusted R-squared:  0.989
## F-statistic: 898.9 on 3 and 27 DF,  p-value: < 2.2e-16
```

To see whether adding the interaction term added anything to the prediction, \marginnote{Though it's hard to imagine being able to improve on an R-squared of 99%.} compare the model with and without using `anova`:

```
anova(pulse.1, pulse.2)
```

```
## Analysis of Variance Table
##
## Model 1: pulse_rate ~ species + temperature
## Model 2: pulse_rate ~ species * temperature
##   Res.Df    RSS Df Sum of Sq     F Pr(>F)
## 1     28 89.350
## 2     27 85.074  1    4.2758 1.357 0.2542
```

There's no significant improvement by adding the interaction, so there's no evidence that having different slopes for each species is necessary. This is the same interpretation as any `anova` for comparing two regressions: the two models are not significantly different in fit, so go with the simpler one, that is, the one without the interaction.

Note that `anova` gave the same P-value as did the *t*-test for the slope coefficient for the interaction in `summary`, 0.254 in both cases. This is because there were only two species and therefore only one slope coefficient was required to distinguish them. If there had been three species, we would have had to look at the `anova` output to hunt for a difference among species, since there would have been two slope coefficients, each with its own P-value.

This wouldn't have told us about the overall effect of species.

If you haven't seen interactions before, don't worry about this. The idea behind it is that we are testing whether we needed lines with different slopes and we concluded that we don't. Don't worry so much about the mechanism behind `pulse.2`; just worry about how it somehow provides a way of modelling two different slopes, one for each species, which we can then test to see whether it helps.

The upshot is that we do not need different slopes; the model `pulse.1` with the same slope for each species describes what is going on.

`ggplot` makes it almost laughably easy to add regression lines for each species to our plot, thus:

```
ggplot(crickets, aes(x = temperature, y = pulse_rate,
    colour = species)) + geom_point() + geom_smooth(method = "lm",
    se = F)
```

The lines are almost exactly parallel, so having the same slope for each species makes perfect sense.

(g) Make suitable residual plots for the regression `pulse.1`.

Solution

First, the plot of residuals against fitted values (after all, it *is* a regression):

```
ggplot(pulse.1, aes(x = .fitted, y = .resid)) +
    geom_point()
```

This looks nice and random.

Now, we plot the residuals against the explanatory variables. There are two, temperature and species, but the latter is categorical. We'll have some extra issues around species, but before we get to that, we have to remember that the data and the output from the regression are in different places when we plot them. There are different ways to get around that. Perhaps the simplest is to use `pulse.1` as our "default" data frame and then get `temperature` from the right place:

```
ggplot(pulse.1, aes(x = crickets$temperature,
    y = .resid)) + geom_point()
```

I don't see anything untoward there.

Species. We want to compare the residuals for the two species, which is categorical. Since the residuals are quantitative, this suggests a boxplot. Remembering to get species from the right place

again, that goes like this:

```
ggplot(pulse.1, aes(x = crickets$species, y = .resid)) +
    geom_boxplot()
```

For the residuals, the median should be zero within each group, and the two groups should be approximately normal with mean 0 and about the same spread. Same spread looks OK, since the boxes are almost exactly the same height, but the normality is not quite there, since both distributions are a little bit skewed to the right. That would also explain why the median residual in each group is a little bit less than zero, because the mathematics requires the overall *mean* residual to be zero, and the right-skewness would make the mean higher than the median.

Is that non-normality really problematic? Well, I could look at the normal quantile plot of all the residuals together:

```
ggplot(pulse.1, aes(sample = .resid)) + stat_qq() +
    stat_qq_line()
```

There's a little weirdness at the top, and a tiny indication of a curve (that would suggest a little right-skewedness), but not really much to worry about. If that third-highest residual were a bit lower (say, 3 rather than 3.5) and maybe if the lowest residual was a bit lower, I don't think we'd have anything to complain about at all.

So, I'm not worried.

13.11 *Roller coasters*

A poll on the Discovery Channel asked people to nominate the best roller-coasters in the United States. We will examine the 10 roller-coasters that received the most votes. Two features of a roller-coaster that are of interest are the distance it drops from start to finish, measured here in feet and the duration of the ride, measured in seconds. Is it true that roller-coasters with a bigger drop also tend to have a longer ride? The data are at link.

Roller-coasters work by gravity, so there must be some drop.

These are not to be confused with what your mom insists that you place between your coffee mug and the table.

(a) Read the data into R and verify that you have a sensible number of rows and columns.

Solution

A `.csv`, so the usual for that:

```
my_url = "http://www.utsc.utoronto.ca/~butler/c32/coasters.csv"
coasters = read_csv(my_url)
```

```
## Parsed with column specification:
## cols(
##   coaster_name = col_character(),
##   state = col_character(),
##   drop = col_integer(),
##   duration = col_integer()
```

```
## )
```

```
coasters
```

```
## # A tibble: 10 x 4
##    coaster_name    state       drop duration
##    <chr>           <chr>      <int>    <int>
##  1 Incredible Hulk Florida      105      135
##  2 Millennium For~ Ohio         300      105
##  3 Goliath         California   255      180
##  4 Nitro           New Jersey   215      240
##  5 Magnum XL-2000  Ohio         195      120
##  6 The Beast       Ohio         141       65
##  7 Son of Beast    Ohio         214      140
##  8 Thunderbolt     Pennsylva~    95       90
##  9 Ghost Rider     California   108      160
## 10 Raven           Indiana       86       90
```

The number of marks for this kind of thing has been decreasing through the course, since by now you ought to have figured out how to do it without looking it up.

There are 10 rows for the promised 10 roller-coasters, and there are several columns: the drop for each roller-coaster and the duration of its ride, as promised, as well as the name of each roller-coaster and the state that it is in. (A lot of them seem to be in Ohio, for some reason that I don't know.) So this all looks good.

(b) Make a scatterplot of duration (response) against drop (explanatory), labelling each roller-coaster with its name in such a way that the labels do not overlap. Add a regression line to your plot.

Solution

The last part, about the labels not overlapping, is an invitation to use `ggrepel`, which is the way I'd recommend doing this. (If not, you have to do potentially lots of work organizing where the labels sit relative to the points, which is time you probably don't want to spend.) Thus:

```
library(ggrepel)
ggplot(coasters, aes(x = drop, y = duration, label = coaster_name)) +
    geom_point() + geom_text_repel() + geom_smooth(method = "lm",
    se = F)
```

The `se=F` at the end is optional; if you omit it, you get that "envelope" around the line, which is fine here.

Note that with the labelling done this way, you can easily identify which roller-coaster is which.

(c) Would you say that roller-coasters with a larger drop tend to have a longer ride? Explain briefly.

Solution

I think there are two good answers here: "yes" and "kind of". Supporting "yes" is the fact that the regression line does go uphill, so that overall, or on average, roller-coasters with a larger drop do tend to have a longer duration of ride as well. Supporting "kind of" is the fact that, though the regression line goes uphill, there are a lot of roller-coasters that are some way off the trend, far from the regression line. I am happy to go with either of those. I could also go with "not really" and the same discussion that I attached to "kind of".

(d) Find a roller-coaster that is unusual compared to the others. What about its combination of `drop` and `duration` is unusual?

Solution

This is an invitation to find a point that is a long way off the line. I think the obvious choice is my first one below, but I would take either of the others as well:

- "Nitro" is a long way above the line. That means it has a long duration, relative to its drop. There are two other roller-coasters that have a larger drop but not as long a duration. In other words, this roller-coaster drops slowly, presumably by doing a lot of twisting, loop-the-loop and so on.

- "The Beast" is a long way below the line, so it has a short duration relative to its drop. It is actually the shortest ride of all, but is only a bit below average in terms of drop. This suggests that The Beast is one of those rides that drops a long way quickly.
- "Millennium Force" has the biggest drop of all, but a shorter-than-average duration. This looks like another ride with a big drop in it.

A roller-coaster that is "unusual" will have a residual that is large in size (either positive, like Nitro, or negative, like the other two). I didn't ask you to find the residuals, but if you want to, `augment` from `broom` is the smoothest way to go:

```
library(broom)
duration.1 = lm(duration ~ drop, data = coasters)
augment(duration.1, coasters) %>% select(coaster_name,
    duration, drop, .resid) %>% arrange(desc(abs(.resid)))
```

```
## # A tibble: 10 x 4
##    coaster_name     duration  drop .resid
##    <chr>               <int> <int>  <dbl>
##  1 Nitro                 240   215  97.0
##  2 The Beast              65   141 -60.1
##  3 Millennium Force      105   300 -58.6
##  4 Ghost Rider           160   108  42.8
##  5 Goliath               180   255  27.3
##  6 Thunderbolt            90    95 -24.0
##  7 Raven                  90    86 -21.8
##  8 Incredible Hulk       135   105  18.6
##  9 Magnum XL-2000        120   195 -18.2
## 10 Son of Beast          140   214  -2.81
```

`augment` produces a data frame (of the original data frame with some new columns that come from the regression), so I can feed it into a pipe to do things with it, like only displaying the columns I want, and arranging them in order by absolute value of residual, so that the roller-coasters further from the line come out first. This identifies the three that we found above. The fourth one, "Ghost Rider", is like Nitro in that it takes a (relatively) long time to fall not very far. You can also put `augment` in the *middle* of a pipe. What you may have to do then is supply the *original* data frame name to `augment` so that you have everything:

```
coasters %>% lm(duration ~ drop, data = .) %>%
    augment(coasters) %>% arrange(desc(abs(.resid)))
```

```
## # A tibble: 10 x 11
##    coaster_name state  drop duration .fitted
##    <chr>        <chr> <int>    <int>   <dbl>
##  1 Nitro        New ~   215      240    143.
##  2 The Beast    Ohio    141       65    125.
##  3 Millennium ~ Ohio    300      105    164.
##  4 Ghost Rider  Cali~   108      160    117.
##  5 Goliath      Cali~   255      180    153.
##  6 Thunderbolt  Penn~    95       90    114.
##  7 Raven        Indi~    86       90    112.
##  8 Incredible ~ Flor~   105      135    116.
##  9 Magnum XL-2~ Ohio    195      120    138.
## 10 Son of Beast Ohio    214      140    143.
## # ... with 6 more variables: .se.fit <dbl>,
## #   .resid <dbl>, .hat <dbl>, .sigma <dbl>,
## #   .cooksd <dbl>, .std.resid <dbl>
```

I wanted to hang on to the roller-coaster names, so I added the data frame name to `augment`. If you don't (that is, you just put `augment()` in the middle of a pipe), then `augment` "attempts to reconstruct the data from the model". That means you wouldn't get *everything* from the original data frame; you would just get the things that were in the regression. In this case, that means you would lose the coaster names.

A quote from the package vignette.

A technicality (but one that you should probably care about): `augment` takes up to *two* inputs: a fitted model object like my `duration.1`, and an optional data frame to include other things from, like the coaster names. I had only one input to it in the pipe because the implied first input was the output from the `lm`, which doesn't have a name; the input `coasters` in the pipe was what would normally be the *second* input to `augment`.

13.12 *Running and blood sugar*

A diabetic wants to know how aerobic exercise affects his blood sugar. When his blood sugar reaches 170 (mg/dl), he goes out for a run at a pace of 10 minutes per mile. He runs different distances on different days. Each time he runs, he measures his blood sugar after the run. (The preferred blood sugar level is between 80 and 120 on this scale.) The data are in the file link. Our aim is to predict blood sugar from distance.

(a) Read in the data and display the data frame that you read in.

Solution

From the URL is easiest. These are delimited by one space, as you can tell by looking at the file:

```
my_url = "http://www.utsc.utoronto.ca/~butler/d29/runner.txt"
runs = read_delim(my_url, " ")
```

```
## Parsed with column specification:
## cols(
##   distance = col_double(),
##   blood_sugar = col_integer()
## )
```

```
runs
```

```
## # A tibble: 12 x 2
##    distance blood_sugar
##       <dbl>       <int>
##  1      2           136
##  2      2           146
##  3      2.5         131
##  4      2.5         125
##  5      3           120
##  6      3           116
##  7      3.5         104
##  8      3.5          95
##  9      4            85
## 10      4            94
## 11      4.5          83
## 12      4.5          75
```

That looks like my data file.

(b) Make a scatterplot and add a smooth trend to it.

Solution

```
ggplot(runs, aes(x = distance, y = blood_sugar)) +
    geom_point() + geom_smooth()
```

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

`blood_sugar` should be on the vertical axis, since this is what we are trying to predict. Getting the `x` and the `y` right is easy on these, because they are the x and y for your plot.

(c) Would you say that the relationship between blood sugar and running distance is approximately linear, or not? It is therefore reasonable to use a regression of blood sugar on distance? Explain briefly.

Solution

I'd say that this is about as linear as you could ever wish for. Neither the pattern of points nor the smooth trend have any kind of noticeable bend in them. (Observing a lack of curvature in either the points or the smooth trend is enough.) The trend is a linear one, so using a regression will be just fine. (If it weren't, the rest of the question would be kind of dumb.)

(d) Fit a suitable regression, and obtain the regression output.

Solution

Two steps: `lm` and then `summary`:

```
runs.1 = lm(blood_sugar ~ distance, data = runs)
summary(runs.1)
```

```
##
## Call:
## lm(formula = blood_sugar ~ distance, data = runs)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -7.8238 -3.6167  0.8333  4.0190  5.5476
```

```
##
## Coefficients:
##             Estimate Std. Error t value
## (Intercept)  191.624      5.439   35.23
## distance     -25.371      1.618  -15.68
##             Pr(>|t|)
## (Intercept) 8.05e-12 ***
## distance    2.29e-08 ***
## ---
## Signif. codes:
##   0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1  ' ' 1
##
## Residual standard error: 4.788 on 10 degrees of freedom
## Multiple R-squared:  0.9609, Adjusted R-squared:  0.957
## F-statistic: 245.7 on 1 and 10 DF,  p-value: 2.287e-08
```

(e) How would you *interpret* the slope? That is, what is the slope, and what does that mean about blood sugar and running distance?

Solution

The slope is -25.37. This means that for each additional mile run, the runner's blood sugar will decrease on average by about 25 units.

You can check this from the scatterplot. For example, from 2 to 3 miles, average blood sugar decreases from about 140 to about 115, a drop of 25.

(f) Is there a (statistically) significant relationship between running distance and blood sugar? How do you know? Do you find this surprising, given what you have seen so far? Explain briefly.

Solution

Look at the P-value either on the `distance` line (for its t-test) or for the F-statistic on the bottom line. These are the same: 0.000000023. (They will be the same any time there is one x-variable.) This P-value is *way* smaller than 0.05, so there *is* a significant relationship between running distance and blood sugar. This does not surprise me in the slightest, because the trend on the scatterplot is *so* clear, there's no way it could have happened by chance if in fact there were no relationship between running distance and blood sugar.

(g) This diabetic is planning to go for a 3-mile run tomorrow and a 5-mile run the day after. Obtain suitable 95% intervals that say what his blood sugar might be after each of these runs.

Solution

This is a prediction interval, in each case, since we are talking about *individual* runs of 3 miles and 5 miles (not the mean blood sugar after *all* runs of 3 miles, which is what a confidence interval for the mean response would be). The procedure is to set up a data frame with the two `distance` values in it, and then feed that and the regression object into `predict`, coming up in a moment.

```
dists = c(3, 5)
dist.new = tibble(distance = dists)
dist.new
```

```
## # A tibble: 2 x 1
##   distance
##      <dbl>
## 1        3
## 2        5
```

The important thing is that the name of the column of the new data frame must be *exactly* the same as the name of the explanatory variable in the regression. If they don't match, `predict` won't work. At least, it won't work properly.

It won't give you an error, but it will go back to the *original* data frame to get distances to predict from, and you will get very confused.

Then, `predict`:

```
pp = predict(runs.1, dist.new, interval = "p")
pp
```

```
##         fit       lwr       upr
## 1 115.50952 104.37000 126.64905
## 2  64.76667  51.99545  77.53788
```

and display this with the distances by the side:

```
cbind(dist.new, pp)
```

```
##   distance       fit       lwr       upr
## 1        3 115.50952 104.37000 126.64905
## 2        5  64.76667  51.99545  77.53788
```

or

```
data.frame(dist.new, pp)
```

```
##   distance       fit       lwr       upr
## 1        3 115.50952 104.37000 126.64905
## 2        5  64.76667  51.99545  77.53788
```

Blood sugar after a 3-mile run is predicted to be between 104 and 127; after a 5-mile run it is predicted to be between 52 and 77.5.

Extra: both `cbind` and `data.frame` are "base R" ways of combining a data frame with something else to make a new data frame. They are not from the `tidyverse`. The `tidyverse` way is via `tibble` or `bind_cols`, but they are a bit more particular about what they will take: `tibble` takes vectors (single variables) and `bind_cols` takes vectors or data frames. The problem here is that `pp` is not either of those:

```
class(pp)
```

```
## [1] "matrix"
```

so that we have to use `as_tibble` first to turn it into a data frame, and thus:

```
pp %>% as_tibble() %>% bind_cols(dist.new)
```

```
## # A tibble: 2 x 4
##     fit   lwr   upr distance
##   <dbl> <dbl> <dbl>    <dbl>
## 1 116.  104.  127.         3
## 2  64.8  52.0  77.5        5
```

which puts things backwards, unless you do it like this:

```
dist.new %>% bind_cols(as_tibble(pp))
```

```
## # A tibble: 2 x 4
##   distance   fit   lwr   upr
##      <dbl> <dbl> <dbl> <dbl>
## 1        3 116.  104.  127.
## 2        5  64.8  52.0  77.5
```

which is a pretty result from very ugly code.

I also remembered that if you finish with a `select`, you get the columns in the order they were in the `select`:

```
pp %>% as_tibble() %>% bind_cols(dist.new) %>%
    select(c(distance, everything()))
```

```
## # A tibble: 2 x 4
##   distance   fit   lwr   upr
##      <dbl> <dbl> <dbl> <dbl>
## 1        3 116.  104.  127.
## 2        5  64.8  52.0  77.5
```

`everything` is a so-called "select helper". It means "everything except any columns you already named", so this whole thing has the effect of listing the columns with `distance` first and all the other columns afterwards, in the order that they were in before.

(h) Which of your two intervals is longer? Does this make sense? Explain briefly.

Solution

The intervals are about 22.25 and 25.5 units long. The one for a 5-mile run is a bit longer. I think this makes sense because 3 miles is close to the average run distance, so there is a lot of "nearby" data. 5 miles is actually longer than any of the runs that were actually done (and therefore we are actually extrapolating), but the important point for the prediction interval is that there is less nearby data: those 2-mile runs don't help so much in predicting blood sugar after a 5-mile run. (They help *some*, because the trend is so linear. This is why the 5-mile interval is not *so* much longer. If the trend were less clear, the 5-mile interval would be more noticeably worse.)

13.13 *Calories and fat in pizza*

The file at link came from a spreadsheet of information about 24 brands of pizza: specifically, per 5-ounce serving, the number of calories, the grams of fat, and the cost (in US dollars). The names of the pizza brands are quite long. This file may open in a spreadsheet when you browse to the link, depending on your computer's setup.

(a) Read in the data and display at least some of the data frame. Are the variables of the right types? (In particular, why is the number of calories labelled one way and the cost labelled a different way?)

Solution

`read_csv` is the thing this time:

```
my_url = "http://www.utsc.utoronto.ca/~butler/d29/Pizza.csv"
pizza = read_csv(my_url)
```

```
## Parsed with column specification:
## cols(
##   Type = col_character(),
##   Calories = col_integer(),
##   Fat = col_double(),
##   Cost = col_double()
## )
```

```
pizza
```

```
## # A tibble: 24 x 4
##    Type                 Calories   Fat  Cost
##    <chr>                   <int> <dbl> <dbl>
```

```
##  1 Domino's Deep Dish ~      385  19.5  1.87
##  2 Pizza Hut's Stuffed~      370  15    1.83
##  3 Pizza Hut's Pan Piz~      280  14    1.83
##  4 Domino's Hand-Tosse~      305  12    1.67
##  5 Pizza Hut's Hand-To~      230   9    1.63
##  6 Little Caesars' Dee~      350  14.2  1.06
##  7 Little Caesars' Ori~      230   8    0.81
##  8 Freschetta Bakes & ~      364  15    0.98
##  9 Freschetta Bakes & ~      334  11    1.23
## 10 DiGiorno Rising Cru~      332  12    0.94
## # ... with 14 more rows
```

The four variables are: the brand of pizza, which got read in as text, the number of calories (an integer), and the fat and cost, which are both decimal numbers so they get labelled `dbl`, which is short for "double-precision floating point number".

Anyway, these are apparently the right thing.

Extra: I wanted to mention something else that I discovered yesterday. There is a package called `rio` that will read (and write) data in a whole bunch of different formats in a unified way. Anyway, the usual installation thing, done once:

R is like that: sometimes it seems as if it has infinite depth.

It does this by figuring what kind of thing you have, from the extension to its filename, and then calling an appropriate function to read in or write out the data. This is an excellent example of *standing on the shoulders of giants* to make our lives easier. The software does the hard work of figuring out what kind of thing you have and how to read it in; all we do is say *import*.

```
install.packages("rio")
```

which takes a moment since it probably has to install some other packages too, and then you read in a file like this:

```
library(rio)
pizza3 = import(my_url)
head(pizza3)
```

```
##                                      Type
## 1        Domino's Deep Dish with Pepperoni
## 2 Pizza Hut's Stuffed Crust with Pepperoni
## 3     Pizza Hut's Pan Pizza with Pepperoni
## 4      Domino's Hand-Tossed with Pepperoni
## 5   Pizza Hut's Hand-Tossed with Pepperoni
## 6 Little Caesars' Deep Dish with Pepperoni
##   Calories  Fat Cost
## 1      385 19.5 1.87
## 2      370 15.0 1.83
## 3      280 14.0 1.83
## 4      305 12.0 1.67
## 5      230  9.0 1.63
## 6      350 14.2 1.06
```

`import` figures that you have a `.csv` file, so it calls up `read_csv` or similar.

Technical note: `rio` does not use the `read_` functions, so what it gives you is actually a `data.frame` rather than a `tibble`, so that when you display it, you get the whole thing even if it is long. Hence the `head` here and below to display the first six lines.

I originally had the data as an Excel spreadsheet, but `import` will gobble up that pizza too:

```
my_other_url = "http://www.utsc.utoronto.ca/~butler/d29/Pizza_E29.xls"
pizza4 = import(my_other_url)
head(pizza4)
```

```
##                                      Type
## 1        Domino's Deep Dish with Pepperoni
## 2 Pizza Hut's Stuffed Crust with Pepperoni
## 3     Pizza Hut's Pan Pizza with Pepperoni
## 4      Domino's Hand-Tossed with Pepperoni
## 5   Pizza Hut's Hand-Tossed with Pepperoni
## 6 Little Caesars' Deep Dish with Pepperoni
##   Calories Fat (g) Cost ($)
## 1      385    19.5     1.87
## 2      370    15.0     1.83
## 3      280    14.0     1.83
## 4      305    12.0     1.67
## 5      230     9.0     1.63
## 6      350    14.2     1.06
```

The corresponding function for writing a data frame to a file in the right format is, predictably enough, called `export`.

(b) Make a scatterplot for predicting calories from the number of grams of fat. Add a smooth trend. What kind of relationship do you see, if any?

Solution

All the variable names start with Capital Letters:

```
ggplot(pizza, aes(x = Fat, y = Calories)) + geom_point() +
    geom_smooth()
```

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

There is definitely an upward trend: the more fat, the more calories. The trend is more or less linear (or, a little bit curved: say what you like, as long as it's not obviously crazy). *I* think, with this much scatter, there's no real justification for fitting a curve.

(c) Fit a straight-line relationship, and display the intercept, slope, R-squared, etc. Is there a real relationship between the two variables, or is any apparent trend just chance?

Solution

`lm`, with `summary`:

```
pizza.1 = lm(Calories ~ Fat, data = pizza)
summary(pizza.1)
```

```
##
## Call:
## lm(formula = Calories ~ Fat, data = pizza)
##
## Residuals:
##    Min     1Q Median     3Q    Max
## -55.44 -11.67   6.18  17.87  41.61
##
## Coefficients:
##             Estimate Std. Error t value
## (Intercept)  194.747     21.605   9.014
## Fat           10.050      1.558   6.449
##             Pr(>|t|)
## (Intercept) 7.71e-09 ***
## Fat         1.73e-06 ***
## ---
## Signif. codes:
```

```
##   0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1  ' ' 1
##
## Residual standard error: 25.79 on 22 degrees of freedom
## Multiple R-squared:  0.654,  Adjusted R-squared:  0.6383
## F-statistic: 41.59 on 1 and 22 DF,  p-value: 1.731e-06
```

To assess whether this trend is real or just chance, look at the P-value on the end of the `Fat` line, or on the bottom line where the F-statistic is (they are the same value of 1.73×10^{-6} or 0.0000017, so you can pick either). This P-value is really small, so the slope is definitely *not* zero, and therefore there really is a relationship between the two variables.

(d) Obtain a plot of the residuals against the fitted values for this regression. Does this indicate that there are any problems with this regression, or not? Explain briefly.

Solution

Use the regression object `pizza.1`:

```
ggplot(pizza.1, aes(x = .fitted, y = .resid)) +
    geom_point() + geom_smooth()
```

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

On my residual plot, I see a slight curve in the smooth trend, but I am not worried about that because the residuals on the plot are all over the place in a seemingly random pattern (the grey envelope is wide and that is pretty close to going straight across). So I think a straight line model is satisfactory.

That's all you needed, but it is also worth looking at a normal quantile plot of the residuals:

```
ggplot(pizza.1, aes(sample = .resid)) + stat_qq() +
    stat_qq_line()
```

A bit skewed to the left (the low ones are too low).

Also a plot of the absolute residuals, for assessing fan-out:

```
ggplot(pizza.1, aes(x = .fitted, y = abs(.resid))) +
    geom_point() + geom_smooth()
```

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

A tiny bit of fan-in (residuals getting *smaller* in size as the fitted value gets bigger), but nothing much, I think.

Another way of assessing curvedness is to fit a squared term anyway, and see whether it is significant:

```
pizza.2 = update(pizza.1, . ~ . + I(Fat^2))
summary(pizza.2)
```

```
##
## Call:
## lm(formula = Calories ~ Fat + I(Fat^2), data = pizza)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -62.103 -14.280   5.513  15.423  35.474
##
## Coefficients:
##             Estimate Std. Error t value
## (Intercept)  90.2544    77.8156   1.160
## Fat          25.9717    11.5121   2.256
## I(Fat^2)     -0.5702     0.4086  -1.395
##             Pr(>|t|)
## (Intercept)   0.2591
## Fat           0.0349 *
## I(Fat^2)      0.1775
## ---
## Signif. codes:
##   0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1  ' ' 1
##
## Residual standard error: 25.25 on 21 degrees of freedom
## Multiple R-squared:  0.6834, Adjusted R-squared:  0.6532
## F-statistic: 22.66 on 2 and 21 DF,  p-value: 5.698e-06
```

The fat-squared term is not significant, so that curve on the smooth trend in the (first) residual plot was indeed nothing to get excited about.

(e) The research assistant in this study returns with two new brands of pizza (ones that were not in the original data). The fat content of a 5-ounce serving was 12 grams for the first brand and 20 grams for the second brand. For each of these brands of pizza, obtain a suitable 95% interval for the number of calories contained in a 5-ounce serving.

Solution

The suitable interval here is a prediction interval, because we are interested in each case in the calorie content of the *particular* pizza brands that the research assistant returned with (and not, for example, in the mean calorie content for *all* brands of pizza that have 12 grams of fat per serving). Thus:

```
newfat = c(12, 20)
new = tibble(Fat = newfat)
new
```

```
## # A tibble: 2 x 1
##     Fat
##   <dbl>
## 1    12
## 2    20
```

```
preds = predict(pizza.1, new, interval = "p")
cbind(new, preds)
```

```
##   Fat      fit      lwr      upr
## 1  12 315.3447 260.5524 370.1369
## 2  20 395.7431 337.1850 454.3011
```

Or, if you like:

```
as_tibble(preds) %>% bind_cols(new) %>% select(Fat,
    everything())
```

```
## # A tibble: 2 x 4
##     Fat   fit   lwr   upr
##   <dbl> <dbl> <dbl> <dbl>
## 1    12  315.  261.  370.
## 2    20  396.  337.  454.
```

For the pizza with 12 grams of fat, the predicted calories are between 261 and 370 with 95% confidence, and for the pizza with 20 grams of fat, the calories are predicted to be between 337 and 454. (You should write down what these intervals are, and not leave the reader to find them in the output.)

(Remember the steps: create a new data frame containing the values to predict for, and then feed that into `predict` along with the model that you want to use to predict with. The variable in the data frame has to be called *precisely* `Fat` with a capital F, otherwise it won't work.)

These intervals are both pretty awful: you get a very weak picture of how many calories per serving the pizza brands in question might contain. This is for two reasons: (i) there was a fair bit of scatter in the original relationship, R-squared being around 65%, and (ii) even if we knew perfectly where the line went (which we don't), there's no guarantee that individual brands of pizza would be on it anyway. (Prediction intervals are always hit by this double whammy, in that individual observations suffer from variability in where the line goes *and* variability around whatever the line is.)

I was expecting, when I put together this question, that the 20-grams-of-fat interval would be noticeably worse, because 20 is farther away from the mean fat content of all the brands. But there isn't

much to choose. For the confidence intervals for the mean calories of *all* brands with these fat contents, the picture is clearer:

```
preds = predict(pizza.1, new, interval = "c")
cbind(new, preds)
```

```
##   Fat      fit      lwr      upr
## 1  12 315.3447 303.4683 327.2211
## 2  20 395.7431 371.9122 419.5739
```

or, as before:

```
as_tibble(preds) %>% bind_cols(new) %>% select(Fat,
    everything())
```

```
## # A tibble: 2 x 4
##     Fat   fit   lwr   upr
##   <dbl> <dbl> <dbl> <dbl>
## 1    12  315.  303.  327.
## 2    20  396.  372.  420.
```

This time the fat-20 interval is noticeably longer than the fat-12 one. And these are much shorter than the prediction intervals, as we would guess.

This question is a fair bit of work for 3 points, so I'm not insisting that you explain your choice of a prediction interval over a confidence interval, but I think it is still a smart thing to do, even purely from a marks point of view, because if you get it wrong for a semi-plausible reason, you might pick up some partial credit. Not pulling out your prediction intervals from your output is a sure way to lose a point, however.

13.14 *Where should the fire stations be?*

In city planning, one major issue is where to locate fire stations. If a city has too many fire stations, it will spend too much on running them, but if it has too few, there may be unnecessary fire damage because the fire trucks take too long to get to the fire.

The first part of a study of this kind of issue is to understand the relationship between the distance from the fire station (measured in miles in our data set) and the amount of fire damage caused (measured in thousands of dollars). A city recorded the fire damage and distance from fire station for 15 residential fires (which you can take as a sample of "all possible residential fires in that city"). The data are in link.

(a) Read in and display the data, verifying that you have the right number of rows and the right columns.

Solution

A quick check of the data reveals that the data values are separated by exactly one space, so:

```
my_url = "http://www.utsc.utoronto.ca/~butler/d29/fire_damage.txt"
fire = read_delim(my_url, " ")
```

```
## Parsed with column specification:
## cols(
##   distance = col_double(),
##   damage = col_double()
## )
```

```
fire
```

```
## # A tibble: 15 x 2
##    distance damage
##       <dbl>  <dbl>
##  1      3.4   26.2
##  2      1.8   17.8
##  3      4.6   31.3
##  4      2.3   23.1
##  5      3.1   27.5
##  6      5.5   36
##  7      0.7   14.1
##  8      3     22.3
##  9      2.6   19.6
## 10      4.3   31.3
## 11      2.1   24
## 12      1.1   17.3
## 13      6.1   43.2
## 14      4.8   36.4
## 15      3.8   26.1
```

15 observations (rows), and promised, and a column each of distances and amounts of fire damage, also as promised.

One mark for reading in the data, and one for saying something convincing about how you have the right thing.

(b)* Obtain a 95% confidence interval for the mean fire damage. (There is nothing here from STAD29, and your answer should have nothing to do with distance.)

Solution

I wanted to dissuade you from thinking too hard here. It's just an ordinary one-sample t-test, extracting the interval from it:

```r
t.test(fire$damage)
```

```
##
##  One Sample t-test
##
## data:  fire$damage
## t = 12.678, df = 14, p-value =
## 4.605e-09
## alternative hypothesis: true mean is not equal to 0
## 95 percent confidence interval:
##  21.94488 30.88178
## sample estimates:
## mean of x
##  26.41333
```

Or

```r
with(fire, t.test(damage))
```

```
##
##  One Sample t-test
##
## data:  damage
## t = 12.678, df = 14, p-value =
## 4.605e-09
## alternative hypothesis: true mean is not equal to 0
## 95 percent confidence interval:
##  21.94488 30.88178
## sample estimates:
## mean of x
##  26.41333
```

Ignore the P-value (it's testing that the mean is the default *zero*, which makes no sense). The confidence interval either way goes from 21.9 to 30.9 (thousand dollars).

(c) Draw a scatterplot for predicting the amount of fire damage from the distance from the fire station. Add a smooth trend to your plot.

Solution
We are predicting fire damage, so that goes on the y-axis:

```r
ggplot(fire, aes(x = distance, y = damage)) +
    geom_point() + geom_smooth()
```

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

(d)* Is there a relationship between distance from fire station and fire damage? Is it linear or definitely curved? How strong is it? Explain briefly.

Solution

When the distance is larger, the fire damage is definitely larger, so there is clearly a relationship. I would call this one approximately linear: it wiggles a bit, but it is not to my mind obviously curved. I would also call it a strong relationship, since the points are close to the smooth trend.

(e) Fit a regression predicting fire damage from distance. How is the R-squared consistent (or inconsistent) with your answer from part~(here)?

Solution

The regression is an ordinary `lm`:

```
damage.1 = lm(damage ~ distance, data = fire)
summary(damage.1)
```

```
##
## Call:
## lm(formula = damage ~ distance, data = fire)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -3.4682 -1.4705 -0.1311  1.7915  3.3915
##
## Coefficients:
##             Estimate Std. Error t value
## (Intercept)  10.2779     1.4203   7.237
## distance      4.9193     0.3927  12.525
```

```
##             Pr(>|t|)
## (Intercept) 6.59e-06 ***
## distance    1.25e-08 ***
## ---
## Signif. codes:
##   0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1  ' ' 1
##
## Residual standard error: 2.316 on 13 degrees of freedom
## Multiple R-squared:  0.9235, Adjusted R-squared:  0.9176
## F-statistic: 156.9 on 1 and 13 DF,  p-value: 1.248e-08
```

We need to display the results, since we need to see the R-squared in order to say something about it.

R-squared is about 92%, high, indicating a strong and linear relationship. Back in part~(here), I said that the relationship is linear and strong, which is entirely consistent with such an R-squared. (If you said something different previously, say how it does or doesn't square with this kind of R-squared value.)

Points: one for fitting the regression, one for displaying it, and two (at the grader's discretion) for saying what the R-squared is and how it's consistent (or not) with part~(here).

Extra: if you thought the trend was "definitely curved", you would find that a parabola (or some other kind of curve) was definitely better than a straight line. Here's the parabola:

```r
damage.2 = lm(damage ~ distance + I(distance^2),
    data = fire)
summary(damage.2)
```

```
##
## Call:
## lm(formula = damage ~ distance + I(distance^2), data = fire)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -2.8856 -1.6915 -0.0179  1.5490  3.6278
##
## Coefficients:
##               Estimate Std. Error t value
## (Intercept)    13.3395     2.5303   5.272
## distance        2.6400     1.6302   1.619
## I(distance^2)   0.3376     0.2349   1.437
##               Pr(>|t|)
## (Intercept)   0.000197 ***
## distance      0.131327
```

```
## I(distance^2) 0.176215
## ---
## Signif. codes:
##   0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1  ' ' 1
##
## Residual standard error: 2.227 on 12 degrees of freedom
## Multiple R-squared:  0.9347, Adjusted R-squared:  0.9238
## F-statistic: 85.91 on 2 and 12 DF,  p-value: 7.742e-08
```

There's no evidence here that a quadratic is better.

Or you might even have thought from the wiggles that it was more like cubic:

```
damage.3 = update(damage.2, . ~ . + I(distance^3))
summary(damage.3)
```

```
##
## Call:
## lm(formula = damage ~ distance + I(distance^2) + I(distance^3),
##     data = fire)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -3.2325 -1.8377  0.0322  1.1512  3.1806
##
## Coefficients:
##               Estimate Std. Error t value
## (Intercept)    10.8466     4.3618   2.487
## distance        5.9555     4.9610   1.200
## I(distance^2)  -0.8141     1.6409  -0.496
## I(distance^3)   0.1141     0.1608   0.709
##               Pr(>|t|)
## (Intercept)     0.0302 *
## distance        0.2552
## I(distance^2)   0.6296
## I(distance^3)   0.4928
## ---
## Signif. codes:
##   0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1  ' ' 1
##
## Residual standard error: 2.274 on 11 degrees of freedom
## Multiple R-squared:  0.9376, Adjusted R-squared:  0.9205
## F-statistic: 55.07 on 3 and 11 DF,  p-value: 6.507e-07
```

No evidence that a cubic is better; that increase in R-squared up to about 94% is just chance (bearing in mind that adding *any* x, even a

useless one, will increase R-squared).

How bendy is the cubic?

```
ggplot(fire, aes(x = distance, y = damage)) +
    geom_point() + geom_smooth(method = "lm") +
    geom_line(data = damage.3, aes(y = .fitted),
        colour = "red")
```

The cubic, in red, does bend a little, but it doesn't do an obvious job of going through the points better than the straight line does. It seems to be mostly swayed by that one observation with damage over 40, and choosing a relationship by how well it fits one point is flimsy at the best of times. So, by Occam's Razor, we go with the line rather than the cubic because it (i) fits equally well, (ii) is simpler.

(f)* Obtain a 95% confidence interval for the mean fire damage *for a residence that is 4 miles from the nearest fire station*. (Note the contrast with part~(here).)

Solution

This is a confidence interval for a mean response at a given value of the explanatory variable. This is as opposed to part~(here), which is averaged over *all* distances. So, follow the steps. Make a tiny data frame with this one value of `distance`:

```
new = tibble(distance = 4)
```

and then feed it into `predict`:

```
pp = predict(damage.1, new, interval = "c")
```

and then put it side-by-side with the value it's a prediction for:

```
cbind(new, pp)
```

```
##   distance      fit      lwr      upr
## 1        4 29.95525 28.52604 31.38446
```

28.5 to 31.4 (thousand dollars).

(g) Compare the confidence intervals of parts (here) and (here). Specifically, compare their centres and their lengths, and explain briefly why the results make sense.

Solution

Let me just put them side by side for ease of comparison: part~(here) is:

```
t.test(fire$damage)
```

```
##
##  One Sample t-test
##
## data:  fire$damage
## t = 12.678, df = 14, p-value =
## 4.605e-09
## alternative hypothesis: true mean is not equal to 0
## 95 percent confidence interval:
##  21.94488 30.88178
## sample estimates:
## mean of x
##  26.41333
```

and part~(here)'s is

```
pp
```

```
##        fit      lwr      upr
## 1 29.95525 28.52604 31.38446
```

I printed them out like this since these give the centres of the intervals as well as the lower and upper limits.

The centre of the interval is higher for the mean damage when the distance is 4. This is because the mean distance is a bit less than 4:

```
fire %>% summarize(m = mean(distance))
```

```
## # A tibble: 1 x 1
##       m
##   <dbl>
## 1  3.28
```

We know it's an upward trend, so our best guess at the mean damage is higher if the mean distance is higher (in (here), the distance is *always* 4: we're looking at the mean fire damage for *all* residences that are 4 miles from a fire station.)

What about the lengths of the intervals? The one in (here) is about $30.9 - 21.9 = 9$ (thousand dollars) long, but the one in (here) is only $31.4 - 28.5 = 2.9$ long, much shorter. This makes sense because the relationship is a strong one: knowing the distance from the fire station is very useful, because the bigger it is, the bigger the damage going to be, with near certainty. Said differently, if you know the distance, you can estimate the damage accurately. If you don't know the distance (as is the case in (here)), you're averaging over a lot of different distances and thus there is a lot of uncertainty in the amount of fire damage also.

If you have some reasonable discussion of the reason why the centres and lengths of the intervals differ, I'm happy. It doesn't have to be the same as mine.

These problems are about multiple regression (more than one x-variable):

13.15 *Being satisfied with hospital*

A hospital administrator collects data to study the effect, if any, of a patient's age, the severity of their illness, and their anxiety level, on the patient's satisfaction with their hospital experience. The data, in the file link, are for 46 patients in a survey. The columns are: patient's satisfaction score `satis`, on a scale of 0 to 100; the patient's `age` (in years), the `severity` of the patient's illness (also on a 0–100 scale), and the patient's `anxiety` score on a standard anxiety test (scale of 0–5). Higher scores mean greater satisfaction, increased severity of illness and more anxiety.

(a) Read in the data and check that you have four columns in your data frame, one for each of your variables.

Solution

This one requires a little thought first. The data values are aligned in columns, and so are the column headers. Thus, `read_table` is what we need:

```
my_url = "http://www.utsc.utoronto.ca/~butler/d29/satisfaction.txt"
satisf = read_table(my_url)
```

```
## Parsed with column specification:
## cols(
```

```
##   satis = col_integer(),
##   age = col_integer(),
##   severity = col_integer(),
##   anxiety = col_double()
## )
```

```
satisf
```

```
## # A tibble: 46 x 4
##    satis   age severity anxiety
##    <int> <int>    <int>   <dbl>
##  1    48    50       51     2.3
##  2    57    36       46     2.3
##  3    66    40       48     2.2
##  4    70    41       44     1.8
##  5    89    28       43     1.8
##  6    36    49       54     2.9
##  7    46    42       50     2.2
##  8    54    45       48     2.4
##  9    26    52       62     2.9
## 10    77    29       50     2.1
## # ... with 36 more rows
```

46 rows and 4 columns: satisfaction score (response), age, severity and anxiety (explanatory).

There is a small question about what to call the data frame. Basically, anything other than `satis` will do, since there will be confusion if your data frame has the same name as one of its columns.

(b)* Obtain scatterplots of the response variable `satis` against each of the other variables.

Solution

The obvious way is to do these one after the other:

```
ggplot(satisf, aes(x = age, y = satis)) + geom_point()
```

```
ggplot(satisf, aes(x = severity, y = satis)) +
    geom_point()
```

```
ggplot(satisf, aes(x = anxiety, y = satis)) +
    geom_point()
```

This is fine, but there is also a way of getting all three plots with *one* `ggplot`. This uses the `facet_wrap` trick, but to set *that* up, we have to have all the x-variables in *one* column, with an extra column labelling which x-variable that value was. This uses `gather`. The right way to do this is in a pipeline:

```
satisf %>% gather(xname, x, age:anxiety) %>% ggplot(aes(x = x,
    y = satis)) + geom_point() + facet_wrap(~xname,
    scales = "free", ncol = 2)
```

Steps: gather together the columns age through anxiety into one column called `x`, with a label in `xname`, then plot this new `x` against satisfaction score, with a separate facet for each different x (in `xname`).

I showed you `facet_grid` in class. What's the difference between that and `facet_wrap`? The difference is that with `facet_wrap`, we are

letting `ggplot` arrange the facets how it wants to. In this case, we didn't care which explanatory variable went on which facet, just as long as we saw all of them somewhere. Inside `facet_wrap` there are *no dots*: a squiggle, followed by the name(s) of the variable(s) that distinguish(es) the facets. The only "design" decision I made here was that the facets should be arranged somehow in two columns, but I didn't care which ones should be where.

If there are more than one, they should be separated by plus signs as in lm. Each facet then has as many labels as variables. I haven't actually done this myself, but from looking at examples, I think this is the way it works.

In `facet_grid`, you have a variable that you want to be displayed in rows or in columns (not just in "different facets"). I'll show you how that works here. Since I am going to draw two plots, I should save the long data frame first and re-use it, rather than calculating it twice (so that I ought now to go back and do the other one using the saved data frame, really):

```
satisf.long = satisf %>% gather(xname, x, age:anxiety)
```

If, at this or any stage, you get confused, the way to un-confuse yourself is to *fire up R Studio and do this yourself*. You have all the data and code you need. If you do it yourself, you can run pipes one line at a time, inspect things, and so on.

First, making a *row* of plots, so that `xname` is the *x* of the facets:

```
ggplot(satisf.long, aes(x = x, y = satis)) + geom_point() +
    facet_grid(. ~ xname, scales = "free")
```

I find these too tall and skinny to see the trends, as on the first `facet_wrap` plot.

And now, making a *column* of plots, with `xname` as *y*:

```
ggplot(satisf.long, aes(x = x, y = satis)) + geom_point() +
    facet_grid(xname ~ ., scales = "free")
```

This one looks weird because the three x-variables are on different scales. The effect of the `scales="free"` is to allow the `satis` scale to vary, but the `x` scale cannot because the facets are all in a line. Compare this:

```
ggplot(satisf.long, aes(x = x, y = satis)) + geom_point() +
    facet_wrap(~xname, ncol = 1, scales = "free")
```

This time, the x scales came out different (and suitable), but I still like squarer plots better for judging relationships.

(c) In your scatterplots of (here), which relationship appears to be the strongest one?

Solution

All the trends appear to be downward ones, but I think `satis` and `age` is the strongest trend. The other ones look more scattered to me.

(d)* Create a correlation matrix for all four variables. Does your strongest trend of the previous part have the strongest correlation?

Solution

This is a matter of running the whole data frame through `cor`:

```
cor(satisf)
```

```
##               satis        age   severity
## satis     1.0000000 -0.7867555 -0.6029417
## age      -0.7867555  1.0000000  0.5679505
## severity -0.6029417  0.5679505  1.0000000
## anxiety  -0.6445910  0.5696775  0.6705287
##             anxiety
## satis    -0.6445910
## age       0.5696775
## severity  0.6705287
## anxiety   1.0000000
```

Ignoring the correlations of variables with themselves, the correlation of `satisf` with `age`, the one I picked out, is the strongest (the most negative trend). If you picked one of the other trends as the strongest, you need to note how close it is to the maximum correlation: for example, if you picked `satis` and `severity`, that's the second highest correlation (in size).

(e) Run a regression predicting satisfaction from the other three variables, and display the output.

Solution

```
satisf.1 = lm(satis ~ age + severity + anxiety,
    data = satisf)
summary(satisf.1)
```

```
##
## Call:
## lm(formula = satis ~ age + severity + anxiety, data = satisf)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -18.3524  -6.4230   0.5196   8.3715  17.1601
##
## Coefficients:
##             Estimate Std. Error t value
## (Intercept) 158.4913    18.1259   8.744
## age          -1.1416     0.2148  -5.315
## severity     -0.4420     0.4920  -0.898
## anxiety     -13.4702     7.0997  -1.897
```

```
##             Pr(>|t|)
## (Intercept) 5.26e-11 ***
## age         3.81e-06 ***
## severity      0.3741
## anxiety       0.0647 .
## ---
## Signif. codes:
##   0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1  ' ' 1
##
## Residual standard error: 10.06 on 42 degrees of freedom
## Multiple R-squared:  0.6822, Adjusted R-squared:  0.6595
## F-statistic: 30.05 on 3 and 42 DF,  p-value: 1.542e-10
```

(f) Does the regression fit well overall? How can you tell?

Solution

For this, look at R-squared, which is 0.682 (68.2%). This is one of those things to have an opinion about. I'd say this is good but not great. I would not call it "poor", since there definitely *is* a relationship, even if it's not a stupendously good one.

(g) Test the null hypothesis that none of your explanatory variables help, against the alternative that one or more of them do. (You'll need an appropriate P-value. Which one is it?) What do you conclude?

Solution

This one is the (global) F-test, whose P-value is at the bottom. It translates to 0.000000000154, so this is *definitely* small, and we reject the null. Thus, one or more of `age`, `severity` and `anxiety` helps to predict satisfaction. (I would like to see this last sentence, rather than just "reject the null".)

(h) The correlation between `severity` and `satis` is not small, but in my regression I found that `severity` was nowhere near significant. Why is this? Explain briefly.

Solution

The key thing to observe is that the t-test in the regression says how important a variable is *given the others that are already in the regression*, or, if you prefer, how much that variable *adds* to the regression, on top of the ones that are already there. So here, we are saying that `severity` has nothing to add, given that the regression already includes the others. (That is, high correlation and strong significance don't always go together.) For a little more insight, look at the correlation matrix of (here) again. The strongest trend with `satis` is with `age`, and indeed `age` is the one obviously significant variable in the regression. The trend of `severity` with `satis` is somewhat downward, and you might otherwise have guessed that this is strong enough to be significant. But see that `severity` *also* has a clear relationship with `age`. A patient with low severity of disease is probably also younger, and we know that younger patients are likely to be more satisfied. Thus severity has nothing (much) to add. The multiple regression is actually doing something clever here. Just looking at the correlations, it appears that all three variables are helpful, but the regression is saying that once you have looked at `age` ("controlled for age"), severity of illness does not have an impact: the correlation of `severity` with `satis` is as big as it is almost entirely because of `age`. This gets into the domain of "partial correlation". If you like videos, you can see link for this. I prefer regression, myself, since I find it clearer. `anxiety` tells a different story: this is close to significant (or *is* significant at the $\alpha = 0.10$ level), so the regression is saying that `anxiety` *does* appear to have something to say about `satis` over and above `age`. This is rather odd, to my mind, since `anxiety` has only a slightly stronger correlation with `satis` and about the same with `age` as `severity` does. But the regression is telling the story to believe, because it handles all the inter-correlations, not just the ones between pairs of variables. I thought it would be rather interesting to do some predictions here. Let's predict satisfaction for all combinations of high and low age, severity and anxiety. I'll use the quartiles for high and low. There is a straightforward but ugly way:

```
quartiles = satisf %>% summarize(age_q1 = quantile(age,
    0.25), age_q3 = quantile(age, 0.75), severity_q1 = quantile(severity,
    0.25), severity_q3 = quantile(severity, 0.75),
    anxiety_q1 = quantile(anxiety, 0.25), anxiety_q3 = quantile(anxiety,
        0.75))
```

This is ugly because of all the repetition (same quantiles of different variables), and the programmer in you should be offended by the ugliness. Anyway, it gives what we want:

```
quartiles
```

```
## # A tibble: 1 x 6
##   age_q1 age_q3 severity_q1 severity_q3
##    <dbl>  <dbl>       <dbl>       <dbl>
## 1   31.2   44.8          48          53
## # ... with 2 more variables:
## #   anxiety_q1 <dbl>, anxiety_q3 <dbl>
```

You can copy the numbers from here to below, or you can do some cleverness to get them in the right places:

```
quartiles %>% gather(var_q, quartile, everything()) %>%
    separate(var_q, c("var_name", "quartile_name")) %>%
    spread(var_name, quartile)
```

```
## Warning: attributes are not identical across measure variables;
## they will be dropped
```

```
## # A tibble: 2 x 4
##   quartile_name   age anxiety severity
##   <chr>         <dbl>   <dbl>    <dbl>
## 1 q1             31.2    2.1        48
## 2 q3             44.8    2.48       53
```

This is what we want for below. Let's test it line by line to see exactly what it did:

```
quartiles %>% gather(var_q, quartile, everything())
```

```
## Warning: attributes are not identical across measure variables;
## they will be dropped
```

```
## # A tibble: 6 x 2
##   var_q       quartile
##   <chr>          <dbl>
## 1 age_q1         31.2
## 2 age_q3         44.8
## 3 severity_q1    48
## 4 severity_q3    53
## 5 anxiety_q1      2.1
## 6 anxiety_q3      2.48
```

Making long format. `everything()` is a select-helper saying "gather up *all* the columns". This is where the (ignorable) warning comes from.

```
quartiles %>% gather(var_q, quartile, everything()) %>%
    separate(var_q, c("var_name", "quartile_name"))
```

```
## Warning: attributes are not identical across measure variables;
## they will be dropped
```

```
## # A tibble: 6 x 3
##   var_name quartile_name quartile
##   <chr>    <chr>            <dbl>
## 1 age      q1               31.2
## 2 age      q3               44.8
## 3 severity q1               48
## 4 severity q3               53
## 5 anxiety  q1                2.1
## 6 anxiety  q3                2.48
```

The column `var_q` above encodes a variable *and* a quartile, so split them up. By default, `separate` splits at an underscore, which is why the things in `quartiles` were named with underscores.

I'd like to claim that I was clever enough to think of this in advance, but I wasn't; originally the variable name and the quartile name were separated by dots, which made the separate more complicated, so I went back and changed it.

```
qq = quartiles %>% gather(var_q, quartile, everything()) %>%
    separate(var_q, c("var_name", "quartile_name")) %>%
    spread(var_name, quartile)
```

```
## Warning: attributes are not identical across measure variables;
## they will be dropped
```

```
qq
```

```
## # A tibble: 2 x 4
##   quartile_name   age anxiety severity
##   <chr>         <dbl>   <dbl>    <dbl>
## 1 q1             31.2    2.1        48
## 2 q3             44.8    2.48       53
```

Last, I want the original variables in columns, so I un-gather to get them there. I spread `var_name`, carrying along the values in `quartile` (which means that the rows will get matched up by `quartile_name`).

Now, let's think about why we were doing that. We want to do predictions of all possible combinations of those values of age and anxiety and severity. Doing "all possible combinations" calls for `crossing`, which looks like this:

```
satisf.new = with(qq, crossing(age, anxiety, severity))
satisf.new
```

```
## # A tibble: 8 x 3
##     age anxiety severity
##   <dbl>   <dbl>    <dbl>
## 1  31.2    2.1        48
```

```
## 2  31.2    2.1        53
## 3  31.2    2.48       48
## 4  31.2    2.48       53
## 5  44.8    2.1        48
## 6  44.8    2.1        53
## 7  44.8    2.48       48
## 8  44.8    2.48       53
```

There are two possibilities for each variable, so there are $2^3 = 8$ "all possible combinations". You can check that `crossing` got them all.

This is a data frame containing all the values we want to predict for, with columns having names that are the same as the variables in the regression, so it's ready to go into `predict`. I'll do prediction intervals, just because:

```
pp = predict(satisf.1, satisf.new, interval = "p")
cbind(satisf.new, pp)
```

```
##     age anxiety severity      fit      lwr
## 1 31.25   2.100       48 73.31233 52.64296
## 2 31.25   2.100       53 71.10231 49.86836
## 3 31.25   2.475       48 68.26102 47.02617
## 4 31.25   2.475       53 66.05100 44.90355
## 5 44.75   2.100       48 57.90057 36.83964
## 6 44.75   2.100       53 55.69055 34.48912
## 7 44.75   2.475       48 52.84926 31.68774
## 8 44.75   2.475       53 50.63924 29.99014
##        upr
## 1 93.98170
## 2 92.33627
## 3 89.49587
## 4 87.19845
## 5 78.96151
## 6 76.89198
## 7 74.01079
## 8 71.28835
```

Looking at the predictions themselves (in `fit`), you can see that `age` has a huge effect. If you compare the 1st and 5nd lines (or the 2nd and 6th, 3rd and 7th, ...) you see that increasing age by 13.5 years, while leaving the other variables the same, decreases the satisfaction score by over 15 on average. Changing `severity`, while leaving everything else the same, has in comparison a tiny effect, just over 2 points. (Compare eg. 1st and 2nd lines.) Anxiety has an

in-between effect: increasing anxiety from 2.1 to 2.475, leaving everything else fixed, decreases satisfaction by about 5 points on average.

I chose the quartiles on purpose: to demonstrate the change in average satisfaction by changing the explanatory variable by an appreciable fraction of its range of values. That is, I changed `severity` by "a fair bit", and still the effect on satisfaction scores was small.

Are any of these prediction intervals longer or shorter? We can calculate how long they are. Look at the predictions:

```
pp
```

```
##        fit      lwr      upr
## 1 73.31233 52.64296 93.98170
## 2 71.10231 49.86836 92.33627
## 3 68.26102 47.02617 89.49587
## 4 66.05100 44.90355 87.19845
## 5 57.90057 36.83964 78.96151
## 6 55.69055 34.48912 76.89198
## 7 52.84926 31.68774 74.01079
## 8 50.63924 29.99014 71.28835
```

This is unfortunately not a data frame:

```
class(pp)
```

```
## [1] "matrix"
```

so we make it one before calculating the lengths. We want `upr` minus `lwr`:

```
pp %>% as_tibble() %>% transmute(pi.length = upr -
    lwr)
```

```
## # A tibble: 8 x 1
##   pi.length
##       <dbl>
## 1      41.3
## 2      42.5
## 3      42.5
## 4      42.3
## 5      42.1
## 6      42.4
## 7      42.3
## 8      41.3
```

Now, I don't want to keep the other stuff from `pp`, so I used `transmute` instead of `mutate`; `transmute` keeps *only* the new variable(s) that I calculate and throws away the others.

Usually you want to keep the other variables around as well, which is why you don't see transmute very often.

Then I put that side by side with the values being predicted for:

```
pp %>% as_tibble() %>% transmute(pi.length = upr -
    lwr) %>% bind_cols(satisf.new)
```

```
## # A tibble: 8 x 4
##   pi.length   age anxiety severity
##       <dbl> <dbl>   <dbl>    <dbl>
## 1      41.3  31.2    2.1        48
## 2      42.5  31.2    2.1        53
## 3      42.5  31.2    2.48       48
## 4      42.3  31.2    2.48       53
## 5      42.1  44.8    2.1        48
## 6      42.4  44.8    2.1        53
## 7      42.3  44.8    2.48       48
## 8      41.3  44.8    2.48       53
```

Two of these are noticeably shorter than the others: the first one and the last one. These are low-everything and high-everything. If you look back at the scatterplot matrix of (here), you'll see that the explanatory variables have positive correlations with each other. This means that when one of them is low, the other ones will tend to be low as well (and correspondingly high with high). That is, most of the data is at or near the low-low-low end or the high-high-high end, and so those values will be easiest to predict for.

I was actually expecting more of an effect, but what I expected is actually there.

In the backward elimination part (coming up), I found that only `age` and `anxiety` had a significant impact on satisfaction, so I can plot these two explanatory variables against each other to see where most of the values are:

```
ggplot(satisf, aes(x = age, y = anxiety)) + geom_point()
```

There is basically *no* data with low age and high anxiety, or with high age and low anxiety, so these combinations will be difficult to predict satisfaction for (and thus their prediction intervals will be longer).

(i) Carry out a backward elimination to determine which of `age`, `severity` and `anxiety` are needed to predict satisfaction. What do you get?

Solution

This means starting with the regression containing all the explanatory variables, which is the one I called `satisf.1`:

```
summary(satisf.1)
```

```
##
## Call:
## lm(formula = satis ~ age + severity + anxiety, data = satisf)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -18.3524  -6.4230   0.5196   8.3715  17.1601
##
## Coefficients:
##             Estimate Std. Error t value
## (Intercept) 158.4913    18.1259   8.744
## age          -1.1416     0.2148  -5.315
## severity     -0.4420     0.4920  -0.898
## anxiety     -13.4702     7.0997  -1.897
##             Pr(>|t|)
## (Intercept) 5.26e-11 ***
## age         3.81e-06 ***
## severity      0.3741
## anxiety       0.0647 .
## ---
## Signif. codes:
##   0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1  ' ' 1
##
## Residual standard error: 10.06 on 42 degrees of freedom
## Multiple R-squared:  0.6822, Adjusted R-squared:  0.6595
## F-statistic: 30.05 on 3 and 42 DF,  p-value: 1.542e-10
```

Pull out the least-significant (highest P-value) variable, which here is `severity`. We already decided that this had nothing to add:

```
satisf.2 = update(satisf.1, . ~ . - severity)
summary(satisf.2)
```

```
##
## Call:
## lm(formula = satis ~ age + anxiety, data = satisf)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -19.4453  -7.3285   0.6733   8.5126  18.0534
##
## Coefficients:
##             Estimate Std. Error t value
## (Intercept) 145.9412    11.5251  12.663
## age          -1.2005     0.2041  -5.882
## anxiety     -16.7421     6.0808  -2.753
##             Pr(>|t|)
## (Intercept) 4.21e-16 ***
## age         5.43e-07 ***
## anxiety      0.00861 **
## ---
## Signif. codes:
##   0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1  ' ' 1
##
## Residual standard error: 10.04 on 43 degrees of freedom
## Multiple R-squared:  0.6761, Adjusted R-squared:  0.661
## F-statistic: 44.88 on 2 and 43 DF,  p-value: 2.98e-11
```

If you like, copy and paste the first `lm`, edit it to get rid of `severity`, and run it again. But when I have a "small change" to make to a model, I like to use `update`.

Having taken `severity` out, `anxiety` has become strongly significant. Since all of the explanatory variables are now significant, this is where we stop. If we're predicting satisfaction, we need to know both a patient's age and their anxiety score: being older or more anxious is associated with a *decrease* in satisfaction.

There is also a function `step` that will do this for you:

```r
step(satisf.1, direction = "backward", test = "F")
```

```
## Start:  AIC=216.18
## satis ~ age + severity + anxiety
##
##            Df Sum of Sq    RSS    AIC
## - severity  1     81.66 4330.5 215.06
## <none>                  4248.8 216.19
## - anxiety   1    364.16 4613.0 217.97
## - age       1   2857.55 7106.4 237.84
```

```
##            F value   Pr(>F)
## - severity  0.8072  0.37407
## <none>
## - anxiety   3.5997  0.06468 .
## - age      28.2471 3.81e-06 ***
## ---
## Signif. codes:
##   0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1  ' ' 1
##
## Step:  AIC=215.06
## satis ~ age + anxiety
##
##           Df Sum of Sq    RSS    AIC F value
## <none>                 4330.5 215.06
## - anxiety  1     763.4 5093.9 220.53  7.5804
## - age      1    3483.9 7814.4 240.21 34.5935
##              Pr(>F)
## <none>
## - anxiety   0.00861 **
## - age     5.434e-07 ***
## ---
## Signif. codes:
##   0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1  ' ' 1
```

```
##
## Call:
## lm(formula = satis ~ age + anxiety, data = satisf)
##
## Coefficients:
## (Intercept)          age      anxiety
##      145.94        -1.20       -16.74
```

with the same result. This function doesn't actually use P-values; instead it uses a thing called AIC. At each step, the variable with the lowest AIC comes out, and when `<none>` bubbles up to the top, that's when you stop. The `test="F"` means "include an *F*-test", but the procedure still uses AIC (it just shows you an *F*-test each time as well). In this case, the other variables were in the same order throughout, but they don't have to be (in the same way that removing one variable from a multiple regression can dramatically change the P-values of the ones that remain). Here, at the first step, `<none>` and `anxiety` were pretty close, but when `severity` came out, taking out nothing was a *lot* better than taking out `anxiety`.

The `test="F"` on the end gets you the P-values. Using the *F*-test is right for regressions; for things like logistic regression that we see

later, `test="Chisq"` is the right one to use.

This is F in quotes, meaning F-test, not F without quotes, meaning FALSE.

13.16 *Handling shipments of chemicals*

The data in link are on shipments of chemicals in drums that arrive at a warehouse. In order, the variables are:

- the number of minutes required to handle the shipment
- the number of drums in the shipment
- the total weight of the shipment, in hundreds of pounds.

(a) The data set has two features: *no* column names, and data aligned in columns (that is, more than one space between data values). Read the data in, giving the columns suitable names. To do this, you may have to consult an appropriate help file, or do some searching, perhaps of one of the other questions on this assignment.

Solution

The alignment of columns means that we need to use `read_table`. Once you've figured *that* out, you can search for help by typing `?read_table` in the Console window (the help will appear bottom right), or you can put the same thing into an R Notebook code chunk, and when you run the chunk, the help will be displayed. (Press control-shift-1 to go back to the notebook.) Or you can Google it, of course. The key observation is that you need to supply some column names in `col_names`, like this:

```
my_url = "http://statweb.lsu.edu/EXSTWeb/StatLab/DataSets/NKNWData/CH06PR09.txt"
cols = c("minutes", "drums", "weight")
chemicals = read_table(my_url, col_names = cols)
```

```
## Parsed with column specification:
## cols(
##   minutes = col_double(),
##   drums = col_double(),
##   weight = col_double()
## )
```

```
chemicals
```

```
## # A tibble: 20 x 3
##    minutes drums weight
##      <dbl> <dbl>  <dbl>
##  1      58     7   5.11
```

```
##  2     152    18  16.7
##  3      41     5   3.2
##  4      93    14   7.03
##  5     101    11  11.0
##  6      38     5   4.04
##  7     203    23  22.1
##  8      78     9   7.03
##  9     117    16  10.6
## 10      44     5   4.76
## 11     121    17  11.0
## 12     112    12   9.51
## 13      50     6   3.79
## 14      82    12   6.45
## 15      48     8   4.6
## 16     127    15  13.9
## 17     140    17  13.0
## 18     155    21  15.2
## 19      39     6   3.64
## 20      90    11   9.57
```

I like to define my URL and column names up front. You can define either of them in the `read_table`, but it makes that line longer. Up to you.

There is no `skip` here, because the data file starts right away with the data and we want to use all the values: we are *adding* names to what's in the data file. If you used `skip`, you will be one observation short all the way through, and your output will be slightly different from mine all the way through.

Use any names you like, but they should resemble what the columns actually represent.

(b) Fit a regression predicting the number of minutes required to handle a shipment from the other two variables. Display the results.

Solution

```r
minutes.1 = lm(minutes ~ drums + weight, data = chemicals)
summary(minutes.1)
```

```
##
## Call:
## lm(formula = minutes ~ drums + weight, data = chemicals)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
```

```
## -8.8353 -3.5591 -0.0533  2.4018 15.1515
##
## Coefficients:
##             Estimate Std. Error t value
## (Intercept)   3.3243     3.1108   1.069
## drums         3.7681     0.6142   6.135
## weight        5.0796     0.6655   7.632
##             Pr(>|t|)
## (Intercept)      0.3
## drums       1.10e-05 ***
## weight      6.89e-07 ***
## ---
## Signif. codes:
##   0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1  ' ' 1
##
## Residual standard error: 5.618 on 17 degrees of freedom
## Multiple R-squared:  0.9869, Adjusted R-squared:  0.9854
## F-statistic: 641.6 on 2 and 17 DF,  p-value: < 2.2e-16
```

(c) Explain carefully but briefly what the slope coefficients for the two explanatory variables represent. Do their signs (positive or negative) make practical sense in the context of handling shipments of chemicals?

Solution

The slope coefficient for `drums` is 3.77; this means that a shipment with one extra drum (but the same total weight) would take on average 3.77 minutes longer to handle. Likewise, the slope coefficient for `weight` is 5.08, so a shipment that weighs 1 hundred more pounds but has the same number of drums will take 5.08 more minutes to handle. Or "each additional drum, all else equal, will take 3.77 more minutes to handle", or similar wording. You have to get at two things: a one-unit increase in the explanatory variable going with a certain increase in the response, and *also* the "all else equal" part. How you say it is up to you, but you need to say it. That was two marks. The third one comes from noting that both slope coefficients are positive, so making a shipment either contain more drums or weigh more makes the handling time longer as well. This makes perfect sense, since either kind of increase would make the shipment more difficult to handle, and thus take longer. I was *not* asking about P-values. There isn't really much to say about those: they're both significant, so the handling time depends on both the total weight and the number of drums. Removing either from the regression would be a mistake.

(d) Obtain plots of residuals against fitted values, residuals against

explanatory variables, and a normal quantile plot of the residuals.

Solution

These are the standard plots from a multiple regression. The second one requires care, but the first and last should be straightforward. Residuals against fitted values:

```
ggplot(minutes.1, aes(x = .fitted, y = .resid)) +
    geom_point()
```

The tricky part about the second one is that the x-values and the residuals come from different data frames, which has to get expressed in the `ggplot`. The obvious way is to do the two plots (one for each explanatory variable) one at a time:

```
ggplot(minutes.1, aes(x = chemicals$drums, y = .resid)) +
    geom_point()
```

and

```
ggplot(minutes.1, aes(x = chemicals$weight, y = .resid)) +
    geom_point()
```

$ %$

What would also work is to make a data frame first with the things to plot:

```
dd = tibble(weight = chemicals$weight, drums = chemicals$drums,
    res = resid(minutes.1))
```

and then:

```
ggplot(dd, aes(x = weight, y = res)) + geom_point()
```

and similarly for `drums`. The `resid` with the model name in brackets seems to be necessary.

Another way to approach this is `augment` from `broom`. That does this:

```r
library(broom)
d = minutes.1 %>% augment(chemicals)
as_tibble(d)
```

```
## # A tibble: 20 x 10
##    minutes drums weight .fitted .se.fit
##  *   <dbl> <dbl>  <dbl>   <dbl>   <dbl>
##  1      58     7   5.11    55.7    1.70
##  2     152    18  16.7    156.     2.47
##  3      41     5   3.2     38.4    2.03
##  4      93    14   7.03    91.8    2.91
##  5     101    11  11.0    101.     2.17
##  6      38     5   4.04    42.7    2.11
##  7     203    23  22.1    202.     3.68
##  8      78     9   7.03    72.9    1.45
##  9     117    16  10.6    118.     2.06
## 10      44     5   4.76    46.3    2.28
## 11     121    17  11.0    123.     2.37
## 12     112    12   9.51    96.8    1.27
## 13      50     6   3.79    45.2    1.87
## 14      82    12   6.45    81.3    2.22
## 15      48     8   4.6     56.8    1.73
## 16     127    15  13.9    130.     2.01
## 17     140    17  13.0    134.     1.75
## 18     155    21  15.2    160.     2.70
## 19      39     6   3.64    44.4    1.88
## 20      90    11   9.57    93.4    1.51
## # ... with 5 more variables: .resid <dbl>,
## #   .hat <dbl>, .sigma <dbl>, .cooksd <dbl>,
## #   .std.resid <dbl>
```

and then you can use `d` as the "base" data frame from which everything comes:

```r
ggplot(d, aes(x = drums, y = .resid)) + geom_point()
```

and

```
ggplot(d, aes(x = weight, y = .resid)) + geom_point()
```

or you can even do that trick to put the two plots on facets:

```
d %>% gather(xname, x, drums:weight) %>% ggplot(aes(x = x,
    y = .resid)) + geom_point() + facet_wrap(~xname)
```

Last, the normal quantile plot:

```
ggplot(minutes.1, aes(sample = .resid)) + stat_qq() +
    stat_qq_line()
```

As a check for the grader, there should be four plots, obtained somehow: residuals against fitted values, normal quantile plot of residuals, residuals against `drums`, residuals against `weight`.

(e) Do you have any concerns, looking at the residual plots? Explain briefly.

Solution

The (only) concern I have, looking at those four plots, is the one very positive residual, the one around 15. Take that away, and I think all of the plots are then acceptable. Alternatively, I will take something like "I have no concerns about the form of the relationship",

saying that the *kind* of model being fitted here is OK (no evidence of non-linearity, fanning out, that kind of stuff). It's up to you to decide whether you think "concerns" extends to outliers, high-influence points, etc. The normal quantile plot reveals that the most negative residual, the one around -9, is in fact almost exactly as negative as you would expect the most negative residual to be, so it is not an outlier at all. The residuals are almost exactly normally distributed, *except* for the most positive one. I don't think you can justify fanning-in, since the evidence for that is mostly from the single point on the right. The other points do not really have residuals closer to zero as you move left to right.

Do *not* be tempted to pick out everything you can think of wrong with these plots. The grader can, and will, take away points if you start naming things that are not concerns.

Extra: what else can I find out about that large-positive-residual point? This is where having the "augmented" data frame is a plus:

```
d %>% filter(.resid > 10)
```

```
## # A tibble: 1 x 10
##   minutes drums weight .fitted .se.fit .resid
##     <dbl> <dbl>  <dbl>   <dbl>   <dbl>  <dbl>
## 1     112    12   9.51    96.8    1.27   15.2
## # ... with 4 more variables: .hat <dbl>,
## #   .sigma <dbl>, .cooksd <dbl>,
## #   .std.resid <dbl>
```

As shown. The predicted number of minutes is 96.8, but the actual number of minutes it took is 112. Hence the residual of 15.2. Can we find "similar" numbers of `drums` and `weight` and compare the `minutes`? Try this:

```
chemicals %>% filter(between(weight, 8, 11), between(drums,
    10, 14))
```

```
## # A tibble: 3 x 3
##   minutes drums weight
##     <dbl> <dbl>  <dbl>
## 1     101    11  11.0
## 2     112    12   9.51
## 3      90    11   9.57
```

You might not have seen `between` before, but it works the way you'd expect. Two other shipments with similar numbers of drums and total weight took around 90–100 minutes to handle, so the 112 does look about 15 minutes too long. This was actually an average-sized shipment:

weight between 8 and 11, for example, returning TRUE or FALSE.

```
library(ggrepel)
d %>% mutate(my_label = ifelse(.resid > 10, "residual +",
    "")) %>% ggplot(aes(x = drums, y = weight,
    colour = minutes, label = my_label)) + geom_point() +
    geom_text_repel()
```

so it's a bit of a mystery why it took so long to handle.

I had some fun with the graph: if you set `colour` equal to a continuous variable (as `minutes` is here), you get a continuous colour scale, by default from dark blue (small) to light blue (large). The number of minutes tends to get larger (lighter) as you go up and to the right with bigger shipments. The point labelled "residual +" is the one with the large residual; it is a noticeably lighter blue than the points around it, meaning that it took longer to handle than those points. I used the trick from C32 to label "some" (here one) of the points: create a new label variable with a `mutate` and an `ifelse`, leaving all of the other labels blank so you don't see them.

The blue colour scheme is a little hard to judge values on. Here's another way to do that:

```
d %>% mutate(my_label = ifelse(.resid > 10, "residual +",
    "")) %>% ggplot(aes(x = drums, y = weight,
    colour = minutes, label = my_label)) + geom_point() +
    geom_text_repel() + scale_colour_gradient(low = "red",
    high = "blue")
```

The labelled point is a little more blue (purplish) than the more clearly red points near it.

The other thing to see is that there is also a positive correlation between the number of drums and the total weight, which is what you'd expect. Unlike with some of our other examples, this wasn't strong enough to cause problems; the separate effects of `drums` and `weight` on `minutes` were distinguishable enough to allow both explanatory variables to have a strongly significant effect on `minutes`.

Post scriptum: the "drums" here are not concert-band-type drums, but something like this:

13.17 *Salaries of mathematicians*

A researcher in a scientific foundation wanted to evaluate the relationship between annual salaries of mathematicians and three explanatory variables:

- an index of work quality
- number of years of experience
- an index of publication success.

The data can be found at link. Data from only a relatively small number of mathematicians were available.

(a) Read in the data and check that you have a sensible number of rows and the right number of columns. (What does "a sensible number of rows" mean here?)

Solution

This is a tricky one. There are aligned columns, but *the column headers are not aligned with them*. Thus `read_table2` is what you need.

```
my_url = "http://www.utsc.utoronto.ca/~butler/d29/mathsal.txt"
salaries = read_table2(my_url)
```

```
## Parsed with column specification:
## cols(
##   salary = col_double(),
##   workqual = col_double(),
##   experience = col_double(),
##   pubsucc = col_double()
## )
```

```
salaries
```

```
## # A tibble: 24 x 4
##    salary workqual experience pubsucc
##     <dbl>    <dbl>      <dbl>   <dbl>
##  1   33.2      3.5          9     6.1
##  2   40.3      5.3         20     6.4
##  3   38.7      5.1         18     7.4
##  4   46.8      5.8         33     6.7
##  5   41.4      4.2         31     7.5
##  6   37.5      6           13     5.9
##  7   39        6.8         25     6
##  8   40.7      5.5         30     4
##  9   30.1      3.1          5     5.8
## 10   52.9      7.2         47     8.3
## # ... with 14 more rows
```

24 observations ("only a relatively small number") and 4 columns, one for the response and one each for the explanatory variables.

Or, if you like,

```
dim(salaries)
```

```
## [1] 24  4
```

for the second part: 24 rows and 4 columns again. I note, with only 24 observations, that we don't really have enough data to investigate the effects of three explanatory variables, but we'll do the best we can. If the pattern, whatever it is, is clear enough, we should be OK.

(b) Make scatterplots of `salary` against each of the three explanatory variables. If you can, do this with *one* `ggplot`.

Solution

The obvious way to do this is as three separate scatterplots, and that will definitely work. But you can do it in one go if you think

about facets, and about having all the x-values in one column (and the names of the x-variables in another column):

```
salaries %>% gather(xname, x, -salary) %>% ggplot(aes(x = x,
    y = salary)) + geom_point() + facet_wrap(~xname,
    ncol = 2, scales = "free")
```

If you don't see how that works, run it yourself, one line at a time.

I was thinking ahead a bit while I was coding that: I wanted the three plots to come out about square, and I wanted the plots to have their own scales. The last thing in the `facet_wrap` does the latter, and arranging the plots in two columns (thinking of the plots as a set of four with one missing) gets them more or less square.

If you don't think of those, try it without, and then fix up what you don't like.

(c) Comment briefly on the direction and strength of each relationship with `salary`.

Solution

To my mind, all of the three relationships are going uphill (that's the "direction" part). `experience` is the strongest, and `pubsucc` looks the weakest (that's the "strength" part). If you want to say there is no relationship with `pubsucc`, that's fine too. This is a judgement call. Note that all the relationships are more or less linear (no obvious curves here). We could also investigate the relationships among the explanatory variables:

```
cor(salaries)
```

```
##               salary  workqual experience
## salary     1.0000000 0.6670958  0.8585582
## workqual   0.6670958 1.0000000  0.4669511
```

```
## experience 0.8585582 0.4669511  1.0000000
## pubsucc    0.5581960 0.3227612  0.2537530
##              pubsucc
## salary     0.5581960
## workqual   0.3227612
## experience 0.2537530
## pubsucc    1.0000000
```

Mentally cut off the first row and column (`salary` is the response). None of the remaining correlations are all that high, so we ought not to have any multicollinearity problems.

(d)* Fit a regression predicting salary from the other three variables, and obtain a `summary` of the results.

Solution

```r
salaries.1 = lm(salary ~ workqual + experience +
    pubsucc, data = salaries)
summary(salaries.1)
```

```
##
## Call:
## lm(formula = salary ~ workqual + experience + pubsucc, data = salaries)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -3.2463 -0.9593  0.0377  1.1995  3.3089
##
## Coefficients:
##             Estimate Std. Error t value
## (Intercept) 17.84693    2.00188   8.915
## workqual     1.10313    0.32957   3.347
## experience   0.32152    0.03711   8.664
## pubsucc      1.28894    0.29848   4.318
##             Pr(>|t|)
## (Intercept) 2.10e-08 ***
## workqual    0.003209 **
## experience  3.33e-08 ***
## pubsucc     0.000334 ***
## ---
## Signif. codes:
##   0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1  ' ' 1
##
## Residual standard error: 1.753 on 20 degrees of freedom
## Multiple R-squared:  0.9109, Adjusted R-squared:  0.8975
## F-statistic: 68.12 on 3 and 20 DF,  p-value: 1.124e-10
```

(e) How can we justify the statement "one or more of the explanatory variables helps to predict salary"? How is this consistent with the value of R-squared?

Solution

"One or more of the explanatory variables helps" is an invitation to consider the (global) F-test for the whole regression. This has the very small P-value of 1.124×10^{-10} (from the bottom line of the output): very small, so one or more of the explanatory variables *does* help, and the statement is correct. The idea that something helps to predict salary suggests (especially with such a small number of observations) that we should have a high R-squared. In this case, R-squared is 0.9109, which is indeed high.

(f) Would you consider removing any of the variables from this regression? Why, or why not?

Solution

Look at the P-values attached to each variable. These are all very small: 0.003, 0.00000003 and 0.0003, way smaller than 0.05. So it would be a mistake to take any, even one, of the variables out: doing so would make the regression much worse. If you need convincing of that, see what happens when we take the variable with the highest P-value out — this is `workqual`:

```
salaries.2 = lm(salary ~ experience + pubsucc,
    data = salaries)
summary(salaries.2)
```

```
##
## Call:
## lm(formula = salary ~ experience + pubsucc, data = salaries)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -5.2723 -0.7865 -0.3983  1.7277  3.2060
##
## Coefficients:
##             Estimate Std. Error t value
## (Intercept) 21.02546    2.14819   9.788
## experience   0.37376    0.04104   9.107
## pubsucc      1.52753    0.35331   4.324
##             Pr(>|t|)
## (Intercept) 2.82e-09 ***
## experience  9.70e-09 ***
## pubsucc        3e-04 ***
```

```
## ---
## Signif. codes:
##   0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1  ' ' 1
##
## Residual standard error: 2.137 on 21 degrees of freedom
## Multiple R-squared:  0.8609, Adjusted R-squared:  0.8477
## F-statistic:    65 on 2 and 21 DF,  p-value: 1.01e-09
```

R-squared has gone down from 91% to 86%: maybe not so much in the grand scheme of things, but it is noticeably less. Perhaps better, since we are comparing models with different numbers of explanatory variables, is to compare the *adjusted* R-squared: this has gone down from 90% to 85%. The fact that this has gone down *at all* is enough to say that taking out `workqual` was a mistake.

Adjusted R-squareds are easier to compare in this context, since you don't have to make a judgement about whether it has changed substantially, whatever you think substantially means.

Another way of seeing whether a variable has anything to add in a regression containing the others is a **partial regression plot**. We take the residuals from `salaries.2` above and plot them against the variable we removed, namely `workqual`. If `workqual` has nothing to add, there will be no pattern; if it *does* have something to add, there will be a trend. Like this. I use `augment` from `broom`:

The residuals have to be the ones from a regression *not* including the x-variable you're testing.

```
library(broom)
salaries.2 %>% augment(salaries) %>% ggplot(aes(x = workqual,
    y = .resid)) + geom_point()
```

This is a mostly straight upward trend. So we need to add a linear term in `workqual` to the regression.

Or not take it out in the first place.

(g) Do you think it would be a mistake to take *both* of `workqual` and `pubsucc` out of the regression? Do a suitable test. Was your guess right?

Solution

I think it would be a big mistake. Taking even one of these variables out of the regression is a bad idea (from the t-tests), so taking out more than one would be a *really* bad idea. To perform a test, fit the model without these two explanatory variables:

```
salaries.3 = lm(salary ~ experience, data = salaries)
```

and then use `anova` to compare the two regressions, smaller model first:

```
anova(salaries.3, salaries.1)
```

```
## Analysis of Variance Table
##
## Model 1: salary ~ experience
## Model 2: salary ~ workqual + experience + pubsucc
##   Res.Df     RSS Df Sum of Sq      F
## 1     22 181.191
## 2     20  61.443  2    119.75 19.489
##      Pr(>F)
## 1
## 2 2.011e-05 ***
## ---
## Signif. codes:
##   0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1  ' ' 1
```

The P-value is extremely small, so the bigger model is definitely better than the smaller one: we really do need all three variables. Which is what we guessed.

(h) Back in part (here), you fitted a regression with all three explanatory variables. By making suitable plots, assess whether there is any evidence that (i) that the linear model should be a curve, (ii) that the residuals are not normally distributed, (iii) that there is "fan-out", where the residuals are getting bigger *in size* as the fitted values get bigger? Explain briefly how you came to your conclusions in each case.

Solution

I intended that (i) should just be a matter of looking at residuals vs. fitted values:

```
ggplot(salaries.1, aes(x = .fitted, y = .resid)) +
    geom_point()
```

There is no appreciable pattern on here, so no evidence of a curve (or apparently of any other problems).

Extra: you might read this that we should check residuals against the x-variables as well, which is a similar trick to the above one for plotting response against each of the explanatories. There is one step first, though: use `augment` from `broom` first to get a dataframe with the original x-variables *and* the residuals in it. The following thus looks rather complicated, and if it confuses you, run the code a piece at a time to see what it's doing:

```
salaries.1 %>% augment(salaries) %>% gather(xname,
    x, workqual:pubsucc) %>% ggplot(aes(x = x,
    y = .resid)) + geom_point() + facet_wrap(~xname,
    scales = "free", ncol = 2)
```

These three residual plots are also pretty much textbook random, so no problems here either.

For (ii), look at a normal quantile plot of the residuals, which is not as difficult as the plot I just did:

```
ggplot(salaries.1, aes(sample = .resid)) + stat_qq() +
    stat_qq_line()
```

That is really pretty good. Maybe the *second* smallest point is a bit far off the line, but otherwise there's nothing to worry about. A quick place to look for problems is the extreme observations, and the largest and smallest residuals are almost exactly the size we'd expect them to be.

Our graph for assessing fan-in or fan-out is to plot the *absolute* values of the residuals against the fitted values. The plot from (i) suggests that we won't have any problems here, but to investigate:

```
ggplot(salaries.1, aes(x = .fitted, y = abs(.resid))) +
    geom_point() + geom_smooth()
```

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

This is pretty nearly straight across. You might think it increases a bit at the beginning, but most of the evidence for that comes from the one observation with fitted value near 30 that happens to have a residual near zero. Conclusions based on one observation are not to be trusted! In summary, I'm happy with this linear multiple regression, and I don't see any need to do anything more with it. I am, however, willing to have some sympathy with opinions that differ from mine, if they are supported by those graphs above.

13.18 *Predicting GPA of computer science students*

The file link contains some measurements of academic achievement for a number of university students studying computer science:

- High school grade point average
- Math SAT score
- Verbal SAT score
- Computer Science grade point average
- Overall university grade point average.

(a) Read in the data and display it (or at least the first ten lines).

Solution
The usual:

```
my_url = "http://www.utsc.utoronto.ca/~butler/d29/gpa.txt"
gpa = read_delim(my_url, " ")
```

```
## Parsed with column specification:
## cols(
##   high_GPA = col_double(),
##   math_SAT = col_integer(),
##   verb_SAT = col_integer(),
##   comp_GPA = col_double(),
##   univ_GPA = col_double()
## )
```

```
gpa
```

```
## # A tibble: 105 x 5
##    high_GPA math_SAT verb_SAT comp_GPA
##       <dbl>    <int>    <int>    <dbl>
##  1     3.45      643      589     3.76
##  2     2.78      558      512     2.87
```

```
## 3     2.52      583      503     2.54
## 4     3.67      685      602     3.83
## 5     3.24      592      538     3.29
## 6     2.1       562      486     2.64
## 7     2.82      573      548     2.86
## 8     2.36      559      536     2.03
## 9     2.42      552      583     2.81
## 10    3.51      617      591     3.41
## # ... with 95 more rows, and 1 more
## #   variable: univ_GPA <dbl>
```

Two SAT scores and three GPAs, as promised.

(b)* Make a scatterplot of high school GPA against university GPA. Which variable should be the response and which explanatory? Explain briefly. Add a smooth trend to your plot.

Solution

High school comes before university, so high school should be explanatory and university should be the response. (High school grades are used as an admission criterion to university, so we would hope they would have some predictive value.)

```
ggplot(gpa, aes(x = high_GPA, y = univ_GPA)) +
    geom_point() + geom_smooth()
```

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

(c) Describe any relationship on your scatterplot: its direction, its strength and its shape. Justify your description briefly.

Solution

Taking these points one at a time:

- direction: upward (a higher high-school GPA generally goes with a higher university GPA as well. Or you can say that the lowest high-school GPAs go with the lowest university GPAs, and high with high, at least most of the time).
- strength: something like moderately strong, since while the trend is upward, there is quite a lot of scatter. (This is a judgement call: something that indicates that you are basing your description on something reasonable is fine.)
- shape: I'd call this "approximately linear" since there is no clear curve here. The smooth trend wiggles a bit, but not enough to make me doubt a straight line.

Looking ahead, I also notice that when high-school GPA is high, university GPA is also consistently high, but when high-school GPA is low, the university GPA is sometimes low and sometimes high, a lot more variable. (This suggests problems with fan-in later.) In a practical sense, what this seems to show is that people who do well at university usually did well in high-school as well, but sometimes their high-school grades were not that good. This is especially true for people with university GPAs around 3.25.

(d)* Fit a linear regression for predicting university GPA from high-school GPA and display the results.

Solution

Just this, therefore:

```r
gpa.1 = lm(univ_GPA ~ high_GPA, data = gpa)
summary(gpa.1)
```

```
##
## Call:
## lm(formula = univ_GPA ~ high_GPA, data = gpa)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -0.69040 -0.11922  0.03274  0.17397  0.91278
##
## Coefficients:
##             Estimate Std. Error t value
## (Intercept)  1.09682    0.16663   6.583
## high_GPA     0.67483    0.05342  12.632
##             Pr(>|t|)
## (Intercept) 1.98e-09 ***
## high_GPA     < 2e-16 ***
## ---
```

```
## Signif. codes:
##   0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1  ' ' 1
##
## Residual standard error: 0.2814 on 103 degrees of freedom
## Multiple R-squared:  0.6077, Adjusted R-squared:  0.6039
## F-statistic: 159.6 on 1 and 103 DF,  p-value: < 2.2e-16
```

Extra: this question goes on too long, so I didn't ask you to look at the residuals from this model, but my comments earlier suggested that we might have had some problems with fanning-in (the variability of predictions getting less as the high-school GPA increases). In case you are interested, I'll look at this here. First, residuals against fitted values:

```
ggplot(gpa.1, aes(x = .fitted, y = .resid)) +
    geom_point() + geom_smooth()
```

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

Is that evidence of a trend in the residuals? Dunno. I'm inclined to call this an "inconsequential wiggle" and say it's OK. Note that the grey envelope includes zero all the way across.

Normal quantile plot of residuals:

```
ggplot(gpa.1, aes(sample = .resid)) + stat_qq() +
    stat_qq_line()
```

A somewhat long-tailed distribution: compared to a normal distribution, the residuals are a bit too big in size, both on the positive and negative end.

The problem I was really worried about was the potential of fanning-in, which we can assess by looking at the absolute residuals:

```
ggplot(gpa.1, aes(x = .fitted, y = abs(.resid))) +
    geom_point() + geom_smooth()
```

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

That is definitely a downward trend in the size of the residuals, and I think I was right to be worried before. The residuals should be of similar size all the way across.

The usual problem of this kind is fanning-*out*, where the residuals get *bigger* in size as the fitted values increase. The bigger values equals more spread is the kind of thing that a transformation like

taking logs will handle: the bigger values are all brought downwards, so they will be both smaller and less variable than they were. This one, though, goes the other way, so the only kind of transformation that might help is one at the other end of the scale (think of the Box-Cox lambda scale), something like maybe reciprocal, $\lambda = -1$ maybe.

The other thought I had was that there is this kind of break around a high-school GPA of 3 (go back to the scatterplot of (here)): when the high-school GPA is higher than 3, the university GPA is very consistent (and shows a clear upward trend), but when the high-school GPA is less than 3, the university GPA is very variable and there doesn't seem to be any trend at all. So maybe two separate analyses would be the way to go.

All right, how does Box-Cox work out here, if at all?

```
library(MASS)
```

```
##
## Attaching package: 'MASS'

## The following object is masked from 'package:dplyr':
##
##     select
```

```
boxcox(univ_GPA ~ high_GPA, data = gpa)
```

It doesn't. All right, that answers *that* question.

When I loaded `MASS`, I also loaded its `select` function, and I might want to use the `tidyverse` `select` function later, and things could get confused. So let's "unload" `MASS` now:

That is what that masked message above was about.

```
detach("package:MASS", unload = T)
```

(e) Two students have been admitted to university. One has a high school GPA of 3.0 and the other a high school GPA of 3.5. Obtain suitable intervals that summarize the GPAs that each of these two students might obtain in university.

Solution

Since we are talking about individual students, rather than the mean of *all* students with these GPAs, prediction intervals are called for. The first step is to make a data frame to predict from. This has to contain columns for all the explanatory variables, just `high_GPA` in this case:

```
gpa.new = tibble(high_GPA = c(3, 3.5))
gpa.new
```

```
## # A tibble: 2 x 1
##   high_GPA
##      <dbl>
## 1      3
## 2      3.5
```

and then feed that into `predict`:

```
preds = predict(gpa.1, gpa.new, interval = "p")
preds
```

```
##        fit      lwr      upr
## 1 3.121313 2.560424 3.682202
## 2 3.458728 2.896105 4.021351
```

and display that side by side with the values it was predicted from:

```
cbind(gpa.new, preds)
```

```
##   high_GPA      fit      lwr      upr
## 1      3.0 3.121313 2.560424 3.682202
## 2      3.5 3.458728 2.896105 4.021351
```

or this way, if you like it better:

```
as_tibble(preds) %>% bind_cols(gpa.new) %>% select(high_GPA,
    everything())
```

```
## # A tibble: 2 x 4
##   high_GPA   fit   lwr   upr
##      <dbl> <dbl> <dbl> <dbl>
## 1      3    3.12  2.56  3.68
## 2      3.5  3.46  2.90  4.02
```

Thus the predicted university GPA for the student with high school GPA 3.0 is between 2.6 and 3.7, and for the student with high school GPA 3.5 is between 2.9 and 4.0. (I think this is a good number

of decimals to give, but in any case, you should actually *say* what the intervals are.)

I observe that these intervals are almost exactly the same length. This surprises me a bit, since I would have said that 3.0 is close to the average high-school GPA and 3.5 is noticeably higher. If that's the case, the prediction interval for 3.5 should be longer (since there is less "nearby data"). Was I right about that?

```
gpa %>% summarize(mean = mean(high_GPA), med = median(high_GPA),
    q1 = quantile(high_GPA, 0.25), q3 = quantile(high_GPA,
        0.75))
```

```
## # A tibble: 1 x 4
##    mean   med    q1    q3
##   <dbl> <dbl> <dbl> <dbl>
## 1  3.08  3.17  2.67  3.48
```

More or less: the mean is close to 3, and 3.5 is close to the third quartile. So the thing about the length of the prediction interval is a bit of a mystery. Maybe it works better for the confidence interval for the mean:

```
preds = predict(gpa.1, gpa.new, interval = "c")
cbind(gpa.new, preds)
```

```
##   high_GPA      fit      lwr      upr
## 1      3.0 3.121313 3.066243 3.176383
## 2      3.5 3.458728 3.388147 3.529309
```

These intervals are a lot shorter, since we are talking about *all* students with the high-school GPAs in question, and we therefore no longer have to worry about variation from student to student (which is considerable). But my supposition about length is now correct: the interval for 3.5, which is further from the mean, is a little longer than the interval for 3.0. Thinking about it, it seems that the individual-to-individual variation, which is large, is dominating things for our prediction interval above.

(f)* Now obtain a regression predicting university GPA from high-school GPA as well as the two SAT scores. Display your results.

Solution

Create a new regression with all the explanatory variables you want in it:

```
gpa.2 = lm(univ_GPA ~ high_GPA + math_SAT + verb_SAT,
    data = gpa)
summary(gpa.2)
```

```
##
## Call:
## lm(formula = univ_GPA ~ high_GPA + math_SAT + verb_SAT, data = gpa)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -0.68186 -0.13189  0.01289  0.16186  0.93994
##
## Coefficients:
##              Estimate Std. Error t value
## (Intercept) 0.5793478  0.3422627   1.693
## high_GPA    0.5454213  0.0850265   6.415
## math_SAT    0.0004893  0.0010215   0.479
## verb_SAT    0.0010202  0.0008123   1.256
##             Pr(>|t|)
## (Intercept)   0.0936 .
## high_GPA     4.6e-09 ***
## math_SAT      0.6330
## verb_SAT      0.2120
## ---
## Signif. codes:
##   0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1  ' ' 1
##
## Residual standard error: 0.2784 on 101 degrees of freedom
## Multiple R-squared:  0.6236, Adjusted R-squared:  0.6124
## F-statistic: 55.77 on 3 and 101 DF,  p-value: < 2.2e-16
```

(g) Test whether adding the two SAT scores has improved the prediction of university GPA. What do you conclude?

Solution

Since we added *two* explanatory variables, the t-tests in `gpa.2` don't apply (they tell us whether we can take out *one* x-variable). We might have some suspicions, but that's all they are. So we have to do `anova`:

```r
anova(gpa.1, gpa.2)
```

```
## Analysis of Variance Table
##
## Model 1: univ_GPA ~ high_GPA
## Model 2: univ_GPA ~ high_GPA + math_SAT + verb_SAT
##   Res.Df    RSS Df Sum of Sq      F Pr(>F)
## 1    103 8.1587
## 2    101 7.8288  2   0.32988 2.1279 0.1244
```

If you put the models the other way around, you'll get a negative F-statistic and degrees of freedom, which doesn't make much sense (although the test will still work).

The null hypothesis here is that the two models fit equally well. Since the P-value is not small, we do not reject that null hypothesis, and therefore we conclude that the two models *do* fit equally well, and therefore we prefer the smaller one, the one that predicts university GPA from just high-school GPA. (Or, equivalently, we conclude that those two SAT scores don't add anything to the prediction of how well a student will do at university, once you know their high-school GPA.)

This might surprise you, given what the SATs are supposed to be *for*. But that's what the data say.

(h) Carry out a backward elimination starting out from your model in part (here). Which model do you end up with? Is it the same model as you fit in (here)?

Solution

In the model of (here), `math_SAT` was the least significant, so that comes out first. (I use `update` but I'm not insisting that you do:)

```
gpa.3 = update(gpa.2, . ~ . - math_SAT)
summary(gpa.3)
```

```
##
## Call:
## lm(formula = univ_GPA ~ high_GPA + verb_SAT, data = gpa)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -0.68430 -0.11268  0.01802  0.14901  0.95239
##
## Coefficients:
##              Estimate Std. Error t value
## (Intercept) 0.6838723  0.2626724   2.604
## high_GPA    0.5628331  0.0765729   7.350
## verb_SAT    0.0012654  0.0006283   2.014
##             Pr(>|t|)
## (Intercept)   0.0106 *
## high_GPA    5.07e-11 ***
## verb_SAT      0.0466 *
## ---
## Signif. codes:
##   0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1  ' ' 1
##
```

```
## Residual standard error: 0.2774 on 102 degrees of freedom
## Multiple R-squared:  0.6227, Adjusted R-squared:  0.6153
## F-statistic: 84.18 on 2 and 102 DF,  p-value: < 2.2e-16
```

Here is where we have to stop, since both high-school GPA and verbal SAT score are significant, and so taking either of them out would be a bad idea. This is a *different* model than the one of (here). This is the case, even though the model with high-school GPA only was not significantly worse than the model containing everything. (This goes to show that model-building doesn't always have nice clear answers.) In the model I called `gpa.2`, neither of the SAT scores were anywhere near significant (considered singly), but as soon as we took out one of the SAT scores (my model `gpa.3`), the other one became significant. This goes to show that you shouldn't take out more than one explanatory variable based on the results of the t-tests, and even if you test to see whether you should have taken out both of the SAT, you won't necessarily get consistent results. Admittedly, it's a close decision whether to keep or remove `verb_SAT`, since its P-value is close to 0.05. The other way of tackling this one is via `step`, which does the backward elimination for you (not that it was much work here):

```r
step(gpa.2, direction = "backward", test = "F")
```

```
## Start:  AIC=-264.6
## univ_GPA ~ high_GPA + math_SAT + verb_SAT
##
##            Df Sum of Sq     RSS     AIC
## - math_SAT  1    0.0178  7.8466 -266.36
## - verb_SAT  1    0.1223  7.9511 -264.97
## <none>                   7.8288 -264.60
## - high_GPA  1    3.1896 11.0184 -230.71
##            F value    Pr(>F)
## - math_SAT  0.2294     0.633
## - verb_SAT  1.5777     0.212
## <none>
## - high_GPA 41.1486 4.601e-09 ***
## ---
## Signif. codes:
##   0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1  ' ' 1
##
## Step:  AIC=-266.36
## univ_GPA ~ high_GPA + verb_SAT
##
##            Df Sum of Sq     RSS     AIC
```

```
## <none>                   7.8466 -266.36
## - verb_SAT  1    0.3121  8.1587 -264.26
## - high_GPA  1    4.1562 12.0028 -223.73
##            F value    Pr(>F)
## <none>
## - verb_SAT  4.0571   0.04662 *
## - high_GPA 54.0268 5.067e-11 ***
## ---
## Signif. codes:
##   0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1  ' ' 1

##
## Call:
## lm(formula = univ_GPA ~ high_GPA + verb_SAT, data = gpa)
##
## Coefficients:
## (Intercept)     high_GPA     verb_SAT
##    0.683872     0.562833     0.001265
```

This gives the same result as we did from our backward elimination. The tables with AIC in them are each step of the elimination, and the variable at the top is the one that comes out next. (When `<none>` gets to the top, that's when you stop.) What happened is that the two SAT scores started off highest, but once we removed `math_SAT`, `verb_SAT` jumped below `<none>` and so the verbal SAT score had to stay.

Both the P-value and the AIC say that the decision about keeping or removing `verb_SAT` is very close.

(i) These students were studying computer science at university. Do you find your backward-elimination result sensible or surprising, given this? Explain briefly.

Solution

I would expect computer science students to be strong students generally, good at math and possibly not so good with words. So it is not surprising that high-school GPA figures into the prediction, but I would expect math SAT score to have an impact also, and it does not. It is also rather surprising that verbal SAT score predicts success at university for these computer science students; you wouldn't think that having better skills with words would be helpful. So I'm surprised. Here, I'm looking for some kind of discussion about what's in the final backward-elimination model, and what you would expect to be true of computer science students. Let's look at the final model from the backward elimination again:

```r
summary(gpa.3)
```

```
##
## Call:
## lm(formula = univ_GPA ~ high_GPA + verb_SAT, data = gpa)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -0.68430 -0.11268  0.01802  0.14901  0.95239
##
## Coefficients:
##              Estimate Std. Error t value
## (Intercept) 0.6838723  0.2626724   2.604
## high_GPA    0.5628331  0.0765729   7.350
## verb_SAT    0.0012654  0.0006283   2.014
##             Pr(>|t|)
## (Intercept)   0.0106 *
## high_GPA    5.07e-11 ***
## verb_SAT      0.0466 *
## ---
## Signif. codes:
##   0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1  ' ' 1
##
## Residual standard error: 0.2774 on 102 degrees of freedom
## Multiple R-squared:  0.6227, Adjusted R-squared:  0.6153
## F-statistic: 84.18 on 2 and 102 DF,  p-value: < 2.2e-16
```

The two slope estimates are both positive, meaning that, all else equal, a higher value for each explanatory variable goes with a higher university GPA. This indicates that a higher verbal SAT score goes with a higher university GPA: this is across all the university courses that a student takes, which you would expect to be math and computer science courses for a Comp Sci student, but might include a few electives or writing courses. Maybe what is happening is that the math SAT score is telling the same story as the high-school GPA for these students, and the verbal SAT score is saying something different. (For example, prospective computer science students are mostly likely to have high math SAT scores, so there's not much information there.)

I think I need to look at the correlations:

```r
cor(gpa)
```

```
##           high_GPA  math_SAT  verb_SAT
## high_GPA 1.0000000 0.7681423 0.7261478
## math_SAT 0.7681423 1.0000000 0.8352272
```

```
## verb_SAT 0.7261478 0.8352272 1.0000000
## comp_GPA 0.7914721 0.6877209 0.6387512
## univ_GPA 0.7795631 0.6627837 0.6503012
##           comp_GPA  univ_GPA
## high_GPA 0.7914721 0.7795631
## math_SAT 0.6877209 0.6627837
## verb_SAT 0.6387512 0.6503012
## comp_GPA 1.0000000 0.9390459
## univ_GPA 0.9390459 1.0000000
```

We'll ignore `comp_GPA` (i) because we haven't been thinking about it and (ii) because it's highly correlated with the university GPA anyway. (There isn't a sense that one of the two university GPAs is explanatory and the other is a response, since students are taking the courses that contribute to them at the same time.)

The highest correlation with university GPA of what remains is high-school GPA, so it's not at all surprising that this features in all our regressions. The correlations between university GPA and the two SAT scores are about equal, so there appears to be no reason to favour one of the SAT scores over the other. But, the two SAT scores are highly correlated with *each other* (0.835), which suggests that if you have one, you don't need the other, because they are telling more or less the same story.

That makes me wonder how a regression with the SAT math score and *not* the SAT verbal score would look:

```r
gpa.4 = lm(univ_GPA ~ high_GPA + math_SAT, data = gpa)
summary(gpa.4)
```

```
##
## Call:
## lm(formula = univ_GPA ~ high_GPA + math_SAT, data = gpa)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -0.68079 -0.12264  0.00741  0.16579  0.90010
##
## Coefficients:
##              Estimate Std. Error t value
## (Intercept) 0.6072916  0.3425047   1.773
## high_GPA    0.5710745  0.0827705   6.899
## math_SAT    0.0012980  0.0007954   1.632
##             Pr(>|t|)
## (Intercept)   0.0792 .
## high_GPA     4.5e-10 ***
```

```
## math_SAT      0.1058
## ---
## Signif. codes:
##   0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1  ' ' 1
##
## Residual standard error: 0.2792 on 102 degrees of freedom
## Multiple R-squared:  0.6177, Adjusted R-squared:  0.6102
## F-statistic:  82.4 on 2 and 102 DF,  p-value: < 2.2e-16
```

Math SAT does not quite significantly add anything to the prediction, which confirms that we do better to use the verbal SAT score (surprising though it seems). Though, the two regressions with the single SAT scores, `gpa.3` and `gpa.4`, have almost the same R-squared values. It's not clear-cut at all. In the end, you have to make a call and stand by it.

14
Dates and times

The usual to begin with:

```
library(tidyverse)
```

```
## -- Attaching packages ---- tidyverse 1.2.1 --
```

```
## v ggplot2 3.1.0     v purrr   0.2.5
## v tibble  1.4.2     v dplyr   0.7.8
## v tidyr   0.8.1     v stringr 1.3.1
## v readr   1.1.1     v forcats 0.3.0
```

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

14.1 Dealing with dates in the Worcester Heart Attack study

The Worcester Heart Attack Study is an ongoing study of heart attacks in the Worcester, MA area. The main purpose of the study is to investigate changes over time in incidence and death rates, and also the use of different treatment approaches. We will be mainly using this data set to investigate data handling and dealing with dates. The data can be found at link.

(a) Read the data into R. The reading-in part is straightforward, but check what type of thing each column is. Is that what it should be?

Solution
This is `read_delim`:

```
my_url = "http://www.utsc.utoronto.ca/~butler/c32/whas500.txt"
whas = read_delim(my_url, " ")
```

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

```
## cols(
##   .default = col_integer(),
##   bmi = col_double(),
##   admitdate = col_character(),
##   disdate = col_character(),
##   fdate = col_character()
## )

## See spec(...) for full column specifications.
```

```
whas
```

```
## # A tibble: 500 x 22
##       id   age gender    hr sysbp diasbp
##    <int> <int>  <int> <int> <int>  <int>
##  1     1    83      0    89   152     78
##  2     2    49      0    84   120     60
##  3     3    70      1    83   147     88
##  4     4    70      0    65   123     76
##  5     5    70      0    63   135     85
##  6     6    70      0    76    83     54
##  7     7    57      0    73   191    116
##  8     8    55      0    91   147     95
##  9     9    88      1    63   209    100
## 10    10    54      0   104   166    106
## # ... with 490 more rows, and 16 more
## #   variables: bmi <dbl>, cvd <int>,
## #   afb <int>, sho <int>, chf <int>,
## #   av3 <int>, miord <int>, mitype <int>,
## #   year <int>, admitdate <chr>,
## #   disdate <chr>, fdate <chr>, los <int>,
## #   dstat <int>, lenfol <int>, fstat <int>
```

To see what type everything is, note that when you display a `tibble`, the type of all the columns on the screen is displayed at the top. Click the little right-arrow to see more columns and to check their type.

All the numbers are properly integer (`int`) or decimal (`dbl`) numbers, but the date columns are `chr` or text. This means that they haven't been read as `Date`s (because they were not in year-month-day order). This is (as we will see) unlike SAS, which determined that they were dates, and even used the first 20 rows of the file to determine what format of dates they were.

(b) The date columns should be R dates. They are not year-month-day, so converting them via `as.Date` (which is what `read_delim`

tries to do) will not work. Load the `lubridate` package, and create new columns in your data frame that are properly dates. Save your data frame, and list it to demonstrate that it worked.

Solution

Load `lubridate` first:

```
library(lubridate)
```

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

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

These dates are day-month-year, so we need `dmy` from `lubridate`:

```
whas2 = whas %>% mutate(admit = dmy(admitdate),
    dis = dmy(disdate), f = dmy(fdate))
glimpse(whas2)
```

```
## Observations: 500
## Variables: 25
## $ id        <int> 1, 2, 3, 4, 5, 6, 7, 8...
## $ age       <int> 83, 49, 70, 70, 70, 70...
## $ gender    <int> 0, 0, 1, 0, 0, 0, 0, 0...
## $ hr        <int> 89, 84, 83, 65, 63, 76...
## $ sysbp     <int> 152, 120, 147, 123, 13...
## $ diasbp    <int> 78, 60, 88, 76, 85, 54...
## $ bmi       <dbl> 25.54051, 24.02398, 22...
## $ cvd       <int> 1, 1, 0, 1, 1, 1, 1, 1...
## $ afb       <int> 1, 0, 0, 0, 0, 0, 0, 0...
## $ sho       <int> 0, 0, 0, 0, 0, 0, 0, 0...
## $ chf       <int> 0, 0, 0, 1, 0, 0, 0, 0...
## $ av3       <int> 0, 0, 0, 0, 0, 1, 0, 0...
## $ miord     <int> 1, 0, 0, 0, 0, 0, 0, 0...
## $ mitype    <int> 0, 1, 1, 1, 1, 0, 1, 1...
## $ year      <int> 1, 1, 1, 1, 1, 1, 1, 1...
## $ admitdate <chr> "13-01-1997", "19-01-1...
## $ disdate   <chr> "18-01-1997", "24-01-1...
## $ fdate     <chr> "31-12-2002", "31-12-2...
## $ los       <int> 5, 5, 5, 10, 6, 1, 5, ...
## $ dstat     <int> 0, 0, 0, 0, 0, 1, 0, 0...
## $ lenfol    <int> 2178, 2172, 2190, 297,...
## $ fstat     <int> 0, 0, 0, 1, 0, 1, 0, 1...
```

```
## $ admit     <date> 1997-01-13, 1997-01-1...
## $ dis       <date> 1997-01-18, 1997-01-2...
## $ f         <date> 2002-12-31, 2002-12-3...
```

There are a lot of columns, so I used `glimpse`. The three new variables we created are at the end of the list. They are correctly `Date`s, and they have the right values, the ones we can see at least.

The indentation is up to you. I think it's nice to make the creations of the three new variables line up. You can also make the opening and closing brackets on the long `mutate` aligned, or you can do as I have done here and put two closing brackets on the end. The rationale for this is that each of the variable definition lines in the `mutate` ends either with a comma or an extra closing bracket, the latter being on the last line. Your choice here is a matter of taste or (in your working life) the coding norms of the team you're working with.

You may have been offended by the repetition above. It so happens that these columns' names all end in `date` and they are the only ones that do, so we can use a "select helper" to select only them, and then submit all of them to a `mutate` via `mutate_at`, which goes like this:

```
whas %>% mutate_at(vars(ends_with("date")), funs(d = dmy)) %>%
    glimpse()
```

```
## Observations: 500
## Variables: 25
## $ id          <int> 1, 2, 3, 4, 5, 6, 7,...
## $ age         <int> 83, 49, 70, 70, 70, ...
## $ gender      <int> 0, 0, 1, 0, 0, 0, 0,...
## $ hr          <int> 89, 84, 83, 65, 63, ...
## $ sysbp       <int> 152, 120, 147, 123, ...
## $ diasbp      <int> 78, 60, 88, 76, 85, ...
## $ bmi         <dbl> 25.54051, 24.02398, ...
## $ cvd         <int> 1, 1, 0, 1, 1, 1, 1,...
## $ afb         <int> 1, 0, 0, 0, 0, 0, 0,...
## $ sho         <int> 0, 0, 0, 0, 0, 0, 0,...
## $ chf         <int> 0, 0, 0, 1, 0, 0, 0,...
## $ av3         <int> 0, 0, 0, 0, 0, 1, 0,...
## $ miord       <int> 1, 0, 0, 0, 0, 0, 0,...
## $ mitype      <int> 0, 1, 1, 1, 1, 0, 1,...
## $ year        <int> 1, 1, 1, 1, 1, 1, 1,...
## $ admitdate   <chr> "13-01-1997", "19-01...
## $ disdate     <chr> "18-01-1997", "24-01...
## $ fdate       <chr> "31-12-2002", "31-12...
## $ los         <int> 5, 5, 5, 10, 6, 1, 5...
## $ dstat       <int> 0, 0, 0, 0, 0, 1, 0,...
```

```
## $ lenfol      <int> 2178, 2172, 2190, 29...
## $ fstat       <int> 0, 0, 0, 1, 0, 1, 0,...
## $ admitdate_d <date> 1997-01-13, 1997-01...
## $ disdate_d   <date> 1997-01-18, 1997-01...
## $ fdate_d     <date> 2002-12-31, 2002-12...
```

One line, as you see, not three. The syntax of this is that we first say which columns we want to mutate. We can use any of the select-helpers for this, including listing the column numbers or names; in this case our date variables all ended with `date`. Then we have to give a function, or more than one function, to mutate them with; in this case we wanted to run all the dates-as-text through `dmy`. Because I said `d=dmy`, it takes the original date variable names, glues an underscore on the end and then the `d` that I said, so we create new variables by those names (at the end of the `glimpse` output). If I had just said `funs(dmy)`, we would have *overwritten* the original values, and `admitdate`, `disdate` and `fdate` would now be `dates`. Losing the original variables would have been OK here, but I wanted to show you how to create new variables.

(c) Create three new variables `diff1, diff2, diff3` that are the numbers of days between each of your dates, and save the data frame in which they have been created. Verify that at least some of them are the same as `los` and `lenfol`.

Solution

I don't know what R's internal storage is for dates (it might be seconds or milliseconds or anything, not necessarily days), so subtracting them requires care; you have to divide by the length of a day (in whatever units), thus:

```
whas3 = whas2 %>% mutate(diff1 = (dis - admit)/ddays(1),
    diff2 = (f - admit)/ddays(1), diff3 = (f -
        dis)/ddays(1))
glimpse(whas3)
```

```
## Observations: 500
## Variables: 28
## $ id        <int> 1, 2, 3, 4, 5, 6, 7, 8...
## $ age       <int> 83, 49, 70, 70, 70, 70...
## $ gender    <int> 0, 0, 1, 0, 0, 0, 0, 0...
## $ hr        <int> 89, 84, 83, 65, 63, 76...
## $ sysbp     <int> 152, 120, 147, 123, 13...
## $ diasbp    <int> 78, 60, 88, 76, 85, 54...
## $ bmi       <dbl> 25.54051, 24.02398, 22...
## $ cvd       <int> 1, 1, 0, 1, 1, 1, 1, 1...
```

```
## $ afb       <int> 1, 0, 0, 0, 0, 0, 0, 0...
## $ sho       <int> 0, 0, 0, 0, 0, 0, 0, 0...
## $ chf       <int> 0, 0, 0, 1, 0, 0, 0, 0...
## $ av3       <int> 0, 0, 0, 0, 0, 1, 0, 0...
## $ miord     <int> 1, 0, 0, 0, 0, 0, 0, 0...
## $ mitype    <int> 0, 1, 1, 1, 1, 0, 1, 1...
## $ year      <int> 1, 1, 1, 1, 1, 1, 1, 1...
## $ admitdate <chr> "13-01-1997", "19-01-1...
## $ disdate   <chr> "18-01-1997", "24-01-1...
## $ fdate     <chr> "31-12-2002", "31-12-2...
## $ los       <int> 5, 5, 5, 10, 6, 1, 5, ...
## $ dstat     <int> 0, 0, 0, 0, 0, 1, 0, 0...
## $ lenfol    <int> 2178, 2172, 2190, 297,...
## $ fstat     <int> 0, 0, 0, 1, 0, 1, 0, 1...
## $ admit     <date> 1997-01-13, 1997-01-1...
## $ dis       <date> 1997-01-18, 1997-01-2...
## $ f         <date> 2002-12-31, 2002-12-3...
## $ diff1     <dbl> 5, 5, 5, 10, 6, 1, 5, ...
## $ diff2     <dbl> 2178, 2172, 2190, 297,...
## $ diff3     <dbl> 2173, 2167, 2185, 287,...
```

The extra `d` on the front of `ddays` indicates that these are what is known to `lubridate` as "durations": a period of time 1 day long that could be any day (as opposed to "June 1, 1970" which is 1 day long, but tied to a particular day).

`los` should be the number of days in hospital, what I calculated as `diff1`, and `lenfol` should be the time from being admitted to last followup, which is my `diff2`. My output from `glimpse` confirms that.

Of course, checking that the first few values match is a nice confirmation, but is not actually a *proof*. For that, we should compare all 500 values, and it would be best to do it in such a way that R is comparing all 500 values for us, since it would be a lot more reliable than the human eye. R has a function `all.equal` which does exactly that. By way of warmup:

```
x = 1:4
y = 1:4
z = c(1, 2, 3, 5)
all.equal(x, y)
```

```
## [1] TRUE
```

```
all.equal(x, z)
```

```
## [1] "Mean relative difference: 0.25"
```

I thought the second one was just going to say `FALSE`, but it gave us a message instead, saying how close `x` and `z` were on average, so that we could decide whether they were close enough to call equal, or, as in this case, not.

Anyway:

```
with(whas3, all.equal(lenfol, diff2))

## [1] TRUE

with(whas3, all.equal(los, diff1))

## [1] TRUE
```

so they really are all equal, all 500 of them.

The computer scientists among you will note that I shouldn't have done this, because *diff1* through *diff3* are double-precision decimal numbers, so I should have tested their equality with *lenfol* and *los* by working out the absolute differences and testing whether they were all *small*. On consulting the help for *all.equal*, though, I find that it *does* work properly, because it actually tests whether the things being compared differ by less than a quantity *tolerance* which defaults to 0.00000015, and if they do it calls them equal. This is all tied in with the difference between integers and decimal numbers as they are represented on a computer: exactly and approximately, respectively. A double-precision number has about 16 significant digits of accuracy; equal things won't have all 16 digits equal, most likely, but they would be expected to have at least 8 of those digits the same. CSCA08 stuff, I imagine. This is where you can casually toss around terms like *machine epsilon*. Oh! I just realized something. You know how very very small P-values are shown in R as *<2.2e-16*? *That's* the machine epsilon. Anything smaller than that is indistinguishable from zero, and you can't have a P-value be *exactly* zero. The default *tolerance* I mentioned above is the square root of this, which is normally used for such things.

(d) Construct side-by-side boxplots of the length of followup by each followup status. You'll need to make sure that the followup status, as it gets fed into `ggplot`, is a `factor`, or, at least, not the number that it is now.

Solution

The easiest way to make a factor is to wrap `fstat`, which is a numeric 0 or 1, in `factor()`:

```
ggplot(whas3, aes(x = factor(fstat), y = lenfol)) +
    geom_boxplot()
```

Or create a factor version of `fstat` first:

```
whas3 %>% mutate(ffstat = factor(fstat)) %>% ggplot(aes(x = ffstat,
    y = lenfol)) + geom_boxplot()
```

I think the second way looks better, because you get a cleaner x-axis on your plot. But if you're doing this for exploration, rather than as something that's going to appear in a report for your boss, the first way is fine.

`ggplot` also treats text stuff as categorical where needed, so this also works:

```
whas3 %>% mutate(cfstat = as.character(fstat)) %>%
    ggplot(aes(x = cfstat, y = lenfol)) + geom_boxplot()
```

14.2 *Growth of Mizuna lettuce seeds*

In 2010, a group of students planted some Mizuna lettuce seeds, and recorded how they grew. The data were saved in an Excel spreadsheet, which is at link. The columns are: the date, the height (in cm) of (I presume) the tallest plant, the amount of water added since

the previous date (ml), the temperature in the container where the seedlings were growing, and any additional notes that the students made (edited for length by me). The top line of the data file is variable names.

(a) Read the spreadsheet data.

Solution

This is `read_excel` from package `readxl`. I'm not sure what will happen to the dates yet. Note that this needs a "local" copy of the spreadsheet (that is, you have to download it and save it on your computer, then upload it to `rstudio.cloud`), possibly using `file.choose` to help R find it. I put my copy in the same project folder as I was working in, so I just need the file name:

```
library(readxl)
mizuna = read_excel("mizuna.xlsx")
mizuna
```

```
## # A tibble: 13 x 5
##    date                height water
##    <dttm>               <dbl> <dbl>
##  1 2010-02-16 00:00:00    0     400
##  2 2010-02-18 00:00:00    0       0
##  3 2010-02-19 00:00:00    0     200
##  4 2010-02-22 00:00:00    3.2   100
##  5 2010-02-23 00:00:00    4.5   100
##  6 2010-02-25 00:00:00    6     100
##  7 2010-02-26 00:00:00    6.5   200
##  8 2010-03-01 00:00:00    9.5   200
##  9 2010-03-03 00:00:00   11.1   200
## 10 2010-03-05 00:00:00   13     250
## 11 2010-03-08 00:00:00   14.5   500
## 12 2010-03-10 00:00:00   16     200
## 13 2010-03-17 00:00:00   18.5   800
## # ... with 2 more variables:
## #   temperature <dbl>, notes <chr>
```

The dates *did* get read properly. `dttm` is "date-time", so I guess it's allowing for the possibility that my dates had times attached as well. Unlike with SAS (as we see later), the years came out right.

(b) Make a plot of `height` against your dates, with the points joined by lines.

Solution

```
ggplot(mizuna, aes(x = date, y = height)) + geom_point() +
    geom_line()
```

(c) Label each point on the plot with the amount of water added up to that point.

Solution

This is `water` again. The way to do this is to load `ggrepel`, then add `geom_text_repel` to the plot, by adding `label=water` to the *original* `aes`:

```
library(ggrepel)
ggplot(mizuna, aes(x = date, y = height, label = water)) +
    geom_point() + geom_line() + geom_text_repel(colour = "red")
```

I made the text red, so that you can see it more easily. It "repels" away from the points, but not from the lines joining them. Which

makes me wonder whether this would work better (I explain `alpha` afterwards):

```
library(ggrepel)
ggplot(mizuna, aes(x = date, y = height, label = water)) +
    geom_point() + geom_line() + geom_label_repel(colour = "red",
    alpha = 0.7)
```

The difference between `text` and `label` is that `text` just uses the text of the variable to mark the point, while `label` puts that text in a box.

I think it works better. You can see where the line goes (under the boxes with the labels in them), but you can see the labels clearly.

What that `alpha` does is to make the thing it's attached to (the labels) partly *transparent*. If you leave it out (try it), the black line disappears completely under the label boxes and you can't see where it goes at all. The value you give for `alpha` says how transparent the thing is, from 1 (not transparent at all) down to 0 (invisible). I first tried 0.3, and you could hardly see the boxes; then I tried 0.7 so that the boxes were a bit more prominent but the lines underneath were still slightly visible, and I decided that this is what I liked. I think making the labels a different colour was a good idea, since that helps to distinguish the number on the label from the line underneath.

You can apply `alpha` to pretty much any `ggplot` thing that might be on top of something else, to make it possible to see what's underneath it. The commonest use for it is if you have a scatterplot with a lot of points; normally, you only see some of the points, because the plot is then a sea of black. But if you make the points partly transparent, you can see more of what's nearby that would otherwise have been hidden.

At some point, I also have to show you folks `jitter`, which plots

in slightly different places points that would otherwise overprint each other exactly, and you wouldn't know how many of them there were, like the outliers on the boxplots of German children near the new airport.

14.3 *Types of childbirth*

Childbirths can be of two types: a "vaginal" birth in which the child is born through the mother's vagina in the normal fashion, and a "cesarean section" where a surgeon cuts through the wall of the mother's abdomen, and the baby is delivered through the incision. Cesarean births are used when there are difficulties in pregnancy or during childbirth that would make a vaginal birth too risky. A hospital kept track of the number of vaginal and Cesarean births for the twelve months of 2012. Of interest is whether the Cesarean rate (the ratio of Cesarean births to all births) was increasing, decreasing or remaining stable over that time. The data may be found at link. The columns are the names of the months (in 2012), the number of cesarean births and the number of vaginal births. (The data are not real, but are typical of the kind of thing you would observe.)

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

Solution

This is a space-delimited text file, which means:

```
my_url = "http://www.utsc.utoronto.ca/~butler/c32/birthtypes.txt"
births = read_delim(my_url, " ")
```

```
## Parsed with column specification:
## cols(
##   month = col_character(),
##   cesarean = col_integer(),
##   vaginal = col_integer()
## )
```

```
births
```

```
## # A tibble: 12 x 3
##    month cesarean vaginal
##    <chr>    <int>   <int>
##  1 Jan         11      68
##  2 Feb          9      63
##  3 Mar         10      72
##  4 Apr         18     105
##  5 May         10      90
```

```
##  6 Jun         10      92
##  7 Jul         11      78
##  8 Aug          9      83
##  9 Sep          9      90
## 10 Oct         15     101
## 11 Nov         12     130
## 12 Dec          8     101
```

Some text and two numbers for each month. Check.

(b) Create a column of actual dates and also a column of cesarean rates, as defined above. Store your new data frame in a variable and display it. For the dates, assume that each date is of the 1st of the month that it belongs to.

Solution

The easiest way is to use `str_c` or `paste` to create a text date with year, month and day in some order, and then to use the appropriate function from `lubridate` to turn that into an actual `date`. If you use `str_c`, you (probably) need the `sep` thing to make sure the values get a space between them; `paste` does this automatically. (The next question is whether `ymd` or whatever can cope without spaces, but I'm not exploring that.) The cesarean rate is `cesarean` divided by `cesarean` plus `vaginal`:

```
library(lubridate)
b2 = births %>% mutate(datestr = str_c("2012",
    month, "1", sep = " ")) %>% mutate(thedate = ymd(datestr)) %>%
    mutate(cesarean_rate = cesarean/(cesarean +
        vaginal))
b2
```

```
## # A tibble: 12 x 6
##    month cesarean vaginal datestr thedate
##    <chr>    <int>   <int> <chr>   <date>
##  1 Jan         11      68 2012 J~ 2012-01-01
##  2 Feb          9      63 2012 F~ 2012-02-01
##  3 Mar         10      72 2012 M~ 2012-03-01
##  4 Apr         18     105 2012 A~ 2012-04-01
##  5 May         10      90 2012 M~ 2012-05-01
##  6 Jun         10      92 2012 J~ 2012-06-01
##  7 Jul         11      78 2012 J~ 2012-07-01
##  8 Aug          9      83 2012 A~ 2012-08-01
##  9 Sep          9      90 2012 S~ 2012-09-01
## 10 Oct         15     101 2012 O~ 2012-10-01
## 11 Nov         12     130 2012 N~ 2012-11-01
```

```
## 12 Dec          8     101 2012 D~ 2012-12-01
## # ... with 1 more variable:
## #   cesarean_rate <dbl>
```

If you don't like that, create columns that contain 2012 and 1 all the way down. If you set a column name equal to a single value, that single value gets repeated the right number of times:

This is an example of R's so-called *recycling rules*.

```
births %>% mutate(year = 2012, day = 1)
```

```
## # A tibble: 12 x 5
##    month cesarean vaginal  year   day
##    <chr>    <int>   <int> <dbl> <dbl>
##  1 Jan         11      68  2012     1
##  2 Feb          9      63  2012     1
##  3 Mar         10      72  2012     1
##  4 Apr         18     105  2012     1
##  5 May         10      90  2012     1
##  6 Jun         10      92  2012     1
##  7 Jul         11      78  2012     1
##  8 Aug          9      83  2012     1
##  9 Sep          9      90  2012     1
## 10 Oct         15     101  2012     1
## 11 Nov         12     130  2012     1
## 12 Dec          8     101  2012     1
```

and then use `unite` as in class. The distinction is that `unite` *only* works on columns. It also "swallows up" the columns that it is made out of; in this case, the original year, month and day disappear:

```
b3 = births %>% mutate(year = 2012, day = 1) %>%
    unite(datestr, year, month, day) %>% mutate(thedate = ymd(datestr)) %>%
    mutate(cesarean_rate = cesarean/(cesarean +
        vaginal))
b3
```

```
## # A tibble: 12 x 5
##    datestr cesarean vaginal thedate
##    <chr>      <int>   <int> <date>
##  1 2012_J~       11      68 2012-01-01
##  2 2012_F~        9      63 2012-02-01
##  3 2012_M~       10      72 2012-03-01
##  4 2012_A~       18     105 2012-04-01
##  5 2012_M~       10      90 2012-05-01
##  6 2012_J~       10      92 2012-06-01
##  7 2012_J~       11      78 2012-07-01
```

```
##  8 2012_A~        9      83 2012-08-01
##  9 2012_S~        9      90 2012-09-01
## 10 2012_O~       15     101 2012-10-01
## 11 2012_N~       12     130 2012-11-01
## 12 2012_D~        8     101 2012-12-01
## # ... with 1 more variable:
## #   cesarean_rate <dbl>
```

I don't mind which order you glue your year, month and day together, as long as you construct the dates with the consistent `lubridate` function.

(c) Plot the cesarean rate against time, with a smooth trend. Do you see an upward trend, a downward trend, no trend, or something else?

Solution

This is a scatterplot with time on the x axis:

```
ggplot(b2, aes(x = thedate, y = cesarean_rate)) +
    geom_point() + geom_smooth()
```

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

I like this better than joining the points by lines, since we already have a trend on the plot, but you can do that in some contrasting way:

```
ggplot(b2, aes(x = thedate, y = cesarean_rate)) +
    geom_point() + geom_line(linetype = "dashed") +
    geom_smooth()
```

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

I see a downward trend. ("A downward trend with a wiggle" if you like.) I didn't ask for any explanation. There is a certain unevenness in the trend of the actual data, but the overall picture appears to be downhill.

(d) Try to summarize the trend you just found with a correlation. What goes wrong? How can you fix it?

Solution

Something like this is the obvious guess:

```
with(b2, cor(thedate, cesarean_rate))
```

```
## Error in cor(thedate, cesarean_rate): 'x' must be numeric
```

This fails because `thedate` is not of itself a number. But lurking in the background is how the date is actually *represented*: as a number of days since Jan 1, 1970. Thus, passing it through `as.numeric` might turn it into that:

```
b5 <- b2 %>% mutate(numeric_date = as.numeric(thedate))
b5
```

```
## # A tibble: 12 x 7
##    month cesarean vaginal datestr thedate
##    <chr>    <int>   <int> <chr>   <date>
##  1 Jan         11      68 2012 J~ 2012-01-01
##  2 Feb          9      63 2012 F~ 2012-02-01
##  3 Mar         10      72 2012 M~ 2012-03-01
##  4 Apr         18     105 2012 A~ 2012-04-01
##  5 May         10      90 2012 M~ 2012-05-01
##  6 Jun         10      92 2012 J~ 2012-06-01
##  7 Jul         11      78 2012 J~ 2012-07-01
```

```
##  8 Aug          9      83 2012 A~ 2012-08-01
##  9 Sep          9      90 2012 S~ 2012-09-01
## 10 Oct         15     101 2012 O~ 2012-10-01
## 11 Nov         12     130 2012 N~ 2012-11-01
## 12 Dec          8     101 2012 D~ 2012-12-01
## # ... with 2 more variables:
## #   cesarean_rate <dbl>, numeric_date <dbl>
```

A little mental calculation suggests that these dates in 2012 are a bit over 40 years, that is $40 \times 365 \simeq 14000$ days, since the "zero" date of Jan 1, 1970, and so it turns out. This suggests that we can calculate a correlation with the numeric dates:

```
with(b5, cor(numeric_date, cesarean_rate))
```

```
## [1] -0.7091219
```

and we can make a test of the null hypothesis that the correlation is zero (against a two-sided alternative) thus:

```
with(b5, cor.test(numeric_date, cesarean_rate))
```

```
##
##  Pearson's product-moment correlation
##
## data:  numeric_date and cesarean_rate
## t = -3.1804, df = 10, p-value =
## 0.009813
## alternative hypothesis: true correlation is not equal to 0
## 95 percent confidence interval:
##  -0.9119078 -0.2280145
## sample estimates:
##        cor
## -0.7091219
```

That downward trend is more than just chance, with a P-value just under 0.01. Having said that, though, if you look at the confidence interval for the correlation, it includes almost all the negative values it could be, so that with only 12 observations we really know very little about the correlation other than that it appears to be negative.

Extra: In practice, you would typically have a much longer time series of measurements than this, such as monthly measurements for several years. In looking at only one year, like we did here, we could get trapped by seasonal effects: for example, cesarean rates might always go down through the year and then jump up again in January. Looking at several years would enable us to disentangle seasonal

effects that happen every year from long-term trends. (As an example of this, think of Toronto snowfall: there is almost always snow in the winter and there is never snow in the summer, a seasonal effect, but in assessing climate change, you want to think about long-term trends in snowfall, after allowing for which month you're looking at.)

14.4 *Wolves and caribou*

In Denali National Park, Alaska, the size of the wolf population depends on the size of the caribou population (since wolves hunt and kill caribou). This is a large national park, so caribou are found in very large herds, so big, in fact, that the well-being of the entire herd is not threatened by wolf attacks. Can the size of the caribou population be used to predict the size of the wolf population? The data can be found at link. The columns are: the date of the survey, the name of the park employee in charge of the survey, the caribou population (in hundreds) and the wolf population (actual count).

In fact, it is believed that wolves help keep caribou herds strong by preventing over-population: that is, the weakest caribou are the ones taken by wolves.

The survey is always taken in the fall, but the date varies.

Counting animals in a region, especially rare, hard-to-find animals, is a whole science in itself. These numbers are probably estimates (with some uncertainty).

(a) Take a look at the data file. How would you describe its format? Read it into R, and check that you got something sensible.

Solution

This looks at first sight as if it's separated by spaces, but most of the data values are separated by *more than one* space. If you look further, you'll see that the values are *lined up in columns*, with the variable names aligned at the top. This is exactly the kind of thing that `read_table` will read. We start with the usual `library(tidyverse)`:

```
library(tidyverse)
my_url = "http://www.utsc.utoronto.ca/~butler/c32/caribou.txt"
denali = read_table(my_url)
```

```
## Parsed with column specification:
## cols(
##   date = col_character(),
##   name = col_character(),
##   caribou = col_integer(),
##   wolf = col_integer()
## )
```

```
denali
```

```
## # A tibble: 7 x 4
##   date       name          caribou  wolf
##   <chr>      <chr>           <int> <int>
## 1 09/01/1995 David S.           30    66
```

```
## 2 09/24/1996 Youngjin K.        34    79
## 3 10/03/1997 Srinivasan M.      27    70
## 4 09/15/1998 Lee Anne J.        25    60
## 5 09/08/1999 Stephanie T.       17    48
## 6 09/03/2000 Angus Mc D.        23    55
## 7 10/06/2001 David S.           20    60
```

That worked: four columns with the right names, and the counts of caribou and wolf are numbers. There are only seven years of surveys; in real-life data you would have more. But the point here is working with dates.

The only (small) weirdness is that the dates are text rather than having been converted into dates. This is because they are not year-month-day, which is the only format that gets automatically converted into dates when read in. (You could use `mdy` from `lubridate` to make them dates.)

Extra: you might have wondered how the names survived, even though they have spaces in them, sometimes more than one. Here's how the file looks:

```
date       name             caribou wolf
09/01/1995 David S.         30       66
09/24/1996 Youngjin K.      34       79
10/03/1997 Srinivasan M.    27       70
09/15/1998 Lee Anne J.      25       60
09/08/1999 Stephanie T.     17       48
09/03/2000 Angus Mc D.      23       55
10/06/2001 David S.         20       60
```

What `read_table` looks for is columns that contain spaces *all the way down*, and separates the values there. For example, between the year of`date` and the first name in `name` there is a space all the way down. After the names and before the caribou counts there are several spaces, and there is one space between the words `caribou` and `wolf` in the header line that goes all the way down. Thus four columns, `date`, `name`, `caribou` and `wolf`. This means that the spaces within the names don't cause any problems at all, since the spaces aren't in the same place in *every* line.

The only way this would fail is if *every* first name had the same number of letters in it; then the space between first name and initial of last name *would* be in the same place in every line.

(b) Create a new data frame where the column labelled `date` is now a genuine R `Date`, using something from `lubridate`.

Solution

What you do is to look at the format of the dates as they are now. They appear to be month-day-year, American style.

Not a surprise since Denali National Park is in Alaska.

Thus the function needed is `mdy`. It doesn't matter whether the months are names or numbers:

```
denali <- denali %>% mutate(date = mdy(date))
denali
```

```
## # A tibble: 7 x 4
##   date       name          caribou  wolf
##   <date>     <chr>           <int> <int>
## 1 1995-09-01 David S.           30    66
## 2 1996-09-24 Youngjin K.        34    79
## 3 1997-10-03 Srinivasan M.      27    70
## 4 1998-09-15 Lee Anne J.        25    60
## 5 1999-09-08 Stephanie T.       17    48
## 6 2000-09-03 Angus Mc D.        23    55
## 7 2001-10-06 David S.           20    60
```

I lived on the edge and overwrote both my column and the whole data frame.

It's actually not *really* living on the edge, because if it doesn't work, you go back and read the data in from the file again.

The dates are displayed in ISO format, year-month-day. You see at the top of the column that they now really *are* dates, not just pieces of text that look like dates.

(c) Create new columns containing the days of the week and the month names for the dates.

Solution

This involves digging in the `lubridate` help to find out how to extract things from a date. It turns out that `wday` extracts the day of the week from a date, by default as a number, and `month` gets the month, also by default as a number:

```
denali %>% mutate(mon = month(date), wd = wday(date))
```

```
## # A tibble: 7 x 6
##   date       name  caribou  wolf   mon    wd
##   <date>     <chr>   <int> <int> <dbl> <dbl>
## 1 1995-09-01 Davi~      30    66     9     6
## 2 1996-09-24 Youn~      34    79     9     3
## 3 1997-10-03 Srin~      27    70    10     6
## 4 1998-09-15 Lee ~      25    60     9     3
## 5 1999-09-08 Step~      17    48     9     4
## 6 2000-09-03 Angu~      23    55     9     1
## 7 2001-10-06 Davi~      20    60    10     7
```

This is not what we wanted, though; we wanted the names of the months and of the days. To fix that, add `label=T` to both functions:

```
denali %>% mutate(mon = month(date, label = T),
    wd = wday(date, label = T))
```

```
## # A tibble: 7 x 6
##   date       name  caribou  wolf mon   wd
##   <date>     <chr>   <int> <int> <ord> <ord>
## 1 1995-09-01 Davi~      30    66 Sep   Fri
## 2 1996-09-24 Youn~      34    79 Sep   Tue
## 3 1997-10-03 Srin~      27    70 Oct   Fri
## 4 1998-09-15 Lee ~      25    60 Sep   Tue
## 5 1999-09-08 Step~      17    48 Sep   Wed
## 6 2000-09-03 Angu~      23    55 Sep   Sun
## 7 2001-10-06 Davi~      20    60 Oct   Sat
```

and that cracks it.

No need to save this data frame anywhere, since we're not using any of this later.

Extra: the `ord` means "ordered factor", which makes sense since these are categorical variables with a natural order. This means that you could do something like counting the number of surveys in each month like this:

```
denali %>% mutate(mon = month(date, label = T),
    wd = wday(date, label = T)) %>% count(mon)
```

```
## # A tibble: 2 x 2
##   mon       n
##   <ord> <int>
## 1 Sep       5
## 2 Oct       2
```

(d) Enough playing around with dates. Make a scatterplot of caribou population (explanatory) against wolf population (response). Do you see any relationship?

Solution

Nothing terribly surprising here:

```
ggplot(denali, aes(x = caribou, y = wolf)) + geom_point()
```

If you like, add a smooth trend to it:

This wiggles more than I would like, with such a small number of observations. Try putting someting like *span=2* in the smooth to make it less wiggly.

```
ggplot(denali, aes(x = caribou, y = wolf)) + geom_point() +
    geom_smooth(se = F)
```

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

This is an upward trend: when one population is large, the other one is large too. This is typical for predator-prey relationships: when there is more to eat (more caribou) the wolf population goes up, and when less, it goes down.

(e) On your plot from the previous part, label each point with the year it belongs to. You can do this in two steps: first make a new column containing just the years, and then use it as labels for the points on the plot.

Solution

I'm going to use `geom_text_repel` for the labels from package `ggrepel`. The year values are gotten using the `lubridate` function `year`:

```
denali %>% mutate(year = year(date)) %>% ggplot(aes(x = caribou,
    y = wolf, label = year)) + geom_point() +
    geom_text_repel()
```

I thought about joining up the points in year order. This is actually *not* `geom_line` as you would have guessed, since what that does is to join points in the order of the variable on the x-axis. To join points in the order that they are in the data (what we want here, because the points are in time order in the data), use instead `geom_path`:

I have to say that I didn't know that until just now.

```
denali %>% mutate(year = year(date)) %>% ggplot(aes(x = caribou,
    y = wolf, label = year)) + geom_point() +
    geom_text_repel() + geom_path()
```

In 1996, both populations were large, and both showed a steady decline until 1999. In 2000 and 2001, both populations seemed to be on the way up again, and you can imagine that in a couple of years, things would go back to about where they were in 1995.

(f) Make a plot of caribou population against time (this is done the obvious way). What seems to be happening to the caribou population over time?

Solution

Make a scatterplot, with the survey date as explanatory variable, and caribou population as response (since time always goes on the x-axis):

```
ggplot(denali, aes(x = date, y = caribou)) + geom_point() +
    geom_line()
```

I used an ordinary `geom_line` this time, to connect neighbouring years, as is often done with a time series. The overall trend is downward, though the 1999 value might be a low from which the population is recovering.

(g) The caribou and wolf populations over time are really "time series". See if you can make a plot of *both* the caribou and wolf populations against time. You can make two y-axes, one for caribou and one for wolf; this will probably require some research on your part to figure out.

Solution

The obvious starting point is to note that both the `caribou` and `wolf` columns are animal populations, just of different animals. One way of plotting both populations is to `gather` them up into one column, and then plot them against time, with the two animals distinguished by colour:

```
denali %>% gather(animal, population, caribou:wolf) %>%
    ggplot(aes(x = date, y = population, colour = animal)) +
    geom_point() + geom_line()
```

This is not quite the story, though, because the caribou and wolf populations are on different scales. The caribou population is numbered in hundreds, while the wolf population is an actual count.

The surveys are late in the year, so the one that is nearly in 1996 is actually the 1995 survey.

What would be nice would be to have a secondary y-axis, so that there were two y-scales, one for each animal. This is very easy to manipulate, though (you can change either scale and get a very different-looking graph), so we ought to be careful.

All right, so let's put the caribou on the left:

```
ggplot(denali, aes(x = date, y = caribou)) + geom_line()
```

Or we can add a `colour` aesthetic to distinguish the caribou from

the wolf populations, that we're going to add in a moment. This looks rather odd at first:

```
ggplot(denali, aes(x = date, y = caribou, colour = "caribou")) +
    geom_line()
```

Now we think about adding the wolf numbers. This is done by adding a second `geom_line`, overriding the `y` and the `colour` to specify that this is wolf now:

```
ggplot(denali, aes(x = date, y = caribou, colour = "caribou")) +
    geom_line() + geom_line(aes(y = wolf, colour = "wolf"))
```

What has happened is that we get lines of different colour for each animal, with a legend. So far so good. The problem is that the wolf numbers are about 2.5 times bigger than the caribou numbers, so that we don't get a good sense of how they go up and down together. If we divided the wolf numbers by 2.5, we would see this better:

Which means, if you stop to think about it, that there are *actually* about 40 times more caribou than wolves.

```
ggplot(denali, aes(x = date, y = caribou, colour = "caribou")) +
    geom_line() + geom_line(aes(y = wolf/2.5,
    colour = "wolf"))
```

Now we get to the secondary y-axis. We want to label this `wolf` and have it reflect that we actually made the graph by dividing the wolf values by 2.5:

```
ggplot(denali, aes(x = date, y = caribou, colour = "caribou")) +
    geom_line() + geom_line(aes(y = wolf/2.5,
    colour = "wolf")) + scale_y_continuous(sec.axis = sec_axis(~./2.5,
    name = "wolf"))
```

Woo, and, very possibly, hoo. I got most of these ideas from link.

Now we see how the populations vary over time, and also that they vary together.

This is about the only double-y-axis setup that I like, with scales chosen so that both the series vary about the same amount. By "dis-

creetly" changing the wolf scale, you could make it look as if one population was much bigger than the other, or varied much more than the other. Lies and statistics.

In my opinion, too many people just plot series against time, possibly with a second y-axis. Variables that vary together, like the wolf and caribou populations here, ought to be plotted *against each other* on a scatterplot, possibly with the time points labelled.

And all too often with Excel (spit).

The ambitious among you may like to compare the graphs here with other predator-prey relationships. If you are of a mathematical bent, you might look into the Lotka-Volterra equations, which is a system of two differential equations describing how changes in one population cause changes in the other population.

15
Functions

```
library(tidyverse)
```

```
## -- Attaching packages ---- tidyverse 1.2.1 --
```

```
## v ggplot2 3.1.0     v purrr   0.2.5
## v tibble  1.4.2     v dplyr   0.7.8
## v tidyr   0.8.1     v stringr 1.3.1
## v readr   1.1.1     v forcats 0.3.0
```

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

15.1 Making some R functions

Let's write some simple R functions to convert temperatures, and later to play with text.

(a) A temperature in Celsius is converted to one in Kelvin by adding 273.15. (A temperature of −273.15 Celsius, 0 Kelvin, is the "absolute zero" temperature that nothing can be colder than.) Write a function called `c_to_k` that converts an input Celsius temperature to one in Kelvin, and test that it works.

Solution

This is mostly an exercise in structuring your function correctly. Let's call the input `C` (uppercase C, since lowercase c has a special meaning to R):

```
c_to_k = function(C) {
    C + 273.15
}
c_to_k(0)
```

```
## [1] 273.15
```

```
c_to_k(20)

## [1] 293.15
```

This is the simplest way to do it: the last line of the function, if calculated but not saved, is the value that gets returned to the outside world. The checks suggest that it worked.

If you're used to Python or similar, you might prefer to calculate the value to be returned and then return it. You can do that in R too:

```
c_to_k = function(C) {
    K = C + 273.15
    return(K)
}
c_to_k(0)

## [1] 273.15

c_to_k(20)

## [1] 293.15
```

That works just as well, and for the rest of this question, you can go either way.

R style is to use the last line of the function for the return value, unless you are jumping out of the function before the end, in which case use *return*.

(b) Write a function to convert a Fahrenheit temperature to Celsius. The way you do that is to subtract 32 and then multiply by 5/9.

Solution

On the model of the previous one, we should call this `f_to_c`. I'm going to return the last line, but you can save the calculated value and return that instead:

```
f_to_c = function(F) {
    (F - 32) * 5/9
}
f_to_c(32)

## [1] 0

f_to_c(50)

## [1] 10

f_to_c(68)

## [1] 20
```

Americans are very good at saying things like "temperatures in the 50s", which don't mean much to me, so I like to have benchmarks to work with: these are the Fahrenheit versions of 0, 10, and 20 Celsius.

Thus "in the 50s" means "between about 10 and 15 Celsius".

(c) *Using the functions you already wrote*, write a function to convert an input Fahrenheit temperature to Kelvin.

Solution

This implies that you can piggy-back on the functions you just wrote, which goes as below. First you convert the Fahrenheit to Celsius, and then you convert *that* to Kelvin. (This is less error-prone than trying to use algebra to get a formula for this conversion and then implementing that):

```
f_to_k = function(F) {
    C = f_to_c(F)
    K = c_to_k(C)
    return(K)
}
f_to_k(32)
```

```
## [1] 273.15
```

```
f_to_k(68)
```

```
## [1] 293.15
```

These check because in Celsius they are 0 and 20 and we found the Kelvin equivalents of those to be these values earlier.

I wrote this one with a `return` because I thought it made the structure clearer: run one function, save the result, run another function, save the result, then return what you've got.

(d) Rewrite your Fahrenheit-to-Celsius convertor to take a suitable default value and check that it works as a default.

Solution

You can choose any default you like. I'll take a default of 68 (what I would call "a nice day"):

```
f_to_c = function(F = 68) {
    (F - 32) * 5/9
}
f_to_c(68)
```

```
## [1] 20
```

```
f_to_c()
```

```
## [1] 20
```

The change is in the top line of the function. You see the result: if we run it without an input, we get the same answer as if the input had been 68.

(e) What happens if you feed your Fahrenheit-to-Celsius convertor a *vector* of Fahrenheit temperatures? What if you use it in a `mutate`?

Solution
Try it and see:

```
temps = seq(30, 80, 10)
temps
```

```
## [1] 30 40 50 60 70 80
```

```
f_to_c(temps)
```

```
## [1] -1.111111  4.444444 10.000000 15.555556
## [5] 21.111111 26.666667
```

Each of the Fahrenheit temperatures gets converted into a Celsius one. This is perhaps more useful in a data frame, thus:

```
tibble(temps = seq(30, 80, 10)) %>% mutate(celsius = f_to_c(temps))
```

```
## # A tibble: 6 x 2
##   temps celsius
##   <dbl>   <dbl>
## 1    30   -1.11
## 2    40    4.44
## 3    50   10
## 4    60   15.6
## 5    70   21.1
## 6    80   26.7
```

All the temperatures are side-by-side with their equivalents.
Here's another way to do the above:

```
temps = seq(30, 80, 10)
temps %>% enframe(value = "fahrenheit") %>% mutate(celsius = f_to_c(temps))
```

```
## # A tibble: 6 x 3
##    name fahrenheit celsius
##   <int>      <dbl>   <dbl>
## 1     1         30   -1.11
```

```
## 2     2         40    4.44
## 3     3         50   10
## 4     4         60   15.6
## 5     5         70   21.1
## 6     6         80   26.7
```

`enframe` creates a two-column data frame out of a vector (like `temps`). A vector can have "names", in which case they'll be used as the `name` column; the values will go in a column called `value` unless you rename it, as I did.

(f) Write another function called `wrap` that takes two arguments: a piece of text called `text`, which defaults to `hello`, and another piece of text called `outside`, which defaults to `*`. The function returns `text` with the text `outside` placed before and after, so that calling the function with the defaults should return `*hello*`. To do this, you can use `str_c` from `stringr` (loaded with the `tidyverse`) which places its text arguments side by side and glues them together into one piece of text. Test your function briefly.

Solution
This:

```
wrap = function(text = "hello", outside = "*") {
    str_c(outside, text, outside)
}
```

I can run this with the defaults:

```
wrap()
```

```
## [1] "*hello*"
```

or with text of my choosing:

```
wrap("goodbye", "_")
```

```
## [1] "_goodbye_"
```

I think that's what I meant by "test briefly".

(g) What happens if you want to change the default `outside` but use the default for `text`? How do you make sure that happens? Explore.

Solution
The obvious thing is this, which doesn't work:

```
wrap("!")
```

```
## [1] "*!*"
```

This takes *text* to be `!`, and `outside` to be the default. How do we get `outside` to be `!` instead? The key is to specify the input by name:

```
wrap(outside = "!")
```

```
## [1] "!hello!"
```

This correctly uses the default for `text`.

If you specify inputs without names, they are taken to be in the order that they appear in the function definition. As soon as they get out of order, which typically happens by using the default for something early in the list, as we did here for `text`, you have to specify names for anything that comes after that. These are the names you put on the function's top line.

You can always use names:

```
wrap(text = "thing", outside = "**")
```

```
## [1] "**thing**"
```

and if you use names, they don't even have to be in order:

```
wrap(outside = "!?", text = "fred")
```

```
## [1] "!?fred!?"
```

(h) What happens if you feed your function `wrap` a vector for either of its arguments? What about if you use it in a `mutate`?

Solution

Let's try:

```
mytext = c("a", "b", "c")
wrap(text = mytext)
```

```
## [1] "*a*" "*b*" "*c*"
```

```
myout = c("*", "!")
wrap(outside = myout)
```

```
## [1] "*hello*" "!hello!"
```

If one of the inputs is a vector, the other one gets "recycled" as many times as the vector is long. What if they're both vectors?

```
mytext2 = c("a", "b", "c", "d")
wrap(mytext2, myout)
```

```
## [1] "*a*" "!b!" "*c*" "!d!"
```

This uses the two inputs in parallel, repeating the short one as needed. But this, though it works, gives a warning:

```
wrap(mytext, myout)
```

```
## Warning in stri_c(..., sep = sep, collapse =
## collapse, ignore_null = TRUE): longer object
## length is not a multiple of shorter object
## length
```

```
## [1] "*a*" "!b!" "*c*"
```

This is because the shorter vector (of length 2 here) doesn't go evenly into the longer one (length 3). It gives a warning because this is probably not what you wanted.

The `mutate` thing is easier, because all the columns in a data frame have to be the same length. `LETTERS` is a vector with the uppercase letters in it:

```
tibble(mytext = LETTERS[1:6], myout = c("*", "**",
    "!", "!!", "_", "__")) %>% mutate(newthing = wrap(mytext,
    myout))
```

```
## # A tibble: 6 x 3
##   mytext myout newthing
##   <chr>  <chr> <chr>
## 1 A      *     *A*
## 2 B      **    **B**
## 3 C      !     !C!
## 4 D      !!    !!D!!
## 5 E      _     _E_
## 6 F      __    __F__
```

15.2 *The Collatz sequence*

The Collatz sequence is a sequence of integers $x_1, x_2, \ldots$ defined in a deceptively simple way: if x_n is the current term of the sequence, then x_{n+1} is defined as $x_n/2$ if x_n is even, and $3x_n + 1$ if x_n is odd. We are interested in understanding how this sequence behaves; for example, what happens to it as n gets large, for different choices of the first term x_1? We will explore this numerically with R; the ambitious among you might like to look into the mathematics of it.

(a) What happens to the sequence when it reaches 4? What would be a sensible way of defining where it ends? Explain briefly.

Solution

When the sequence reaches 4 (that is, when its current term is 4), the next term is 2 and the one after that is 1. Then the following term is 4 again ($(3 \times 1) + 1$) and then it repeats indefinitely, $4, 2, 1, 4, 2, 1, \ldots$. I think a sensible way to define where the sequence ends is to say "when it reaches 1", since if you start at 2 you'll never reach 4 (so "when it reaches 4" won't work), and it seems at least plausible that it will hit the cycle 4, 2, 1 sometime.

(b) Write an R function called `is_odd` that returns `TRUE` if its input is an odd number and `FALSE` if it is even (you can assume that the input is an integer and not a decimal number). To do *that*, you can use the function `%%` where `a %% b` is the remainder when `a` is divided by `b`. To think about oddness or evenness, consider the remainder when you divide by 2.

Solution

Let's try this out. For example, 5 is odd and 6 is even, so

```
5%%2
```

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

```
6%%2
```

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

When a number is odd, its remainder on dividing by 2 is 1, and when even, the remainder is 0. There is an additional shortcut here in that 1 is the numeric value of `TRUE` and 0 of `FALSE`, so all we have to do is calculate the remainder on dividing by 2, turn it into a `logical`, and return it:

```
is_odd = function(x) {
    r = x%%2
    as.logical(r)
}
```

You probably haven't seen `as.logical` before, but it's the same idea as `as.numeric`: turn something that looks like a `TRUE` or `FALSE` into something that actually *is*.

We should test it:

```
is_odd(19)
```

```
## [1] TRUE
```

```
is_odd(12)
```

```
## [1] FALSE
```

```
is_odd(0)
```

```
## [1] FALSE
```

0 is usually considered an even number, so this is good.

(c) Write an R function called `hotpo1` that takes an integer as input and returns the next number in the Collatz sequence. To do this, use the function you just wrote that determines whether a number is even or odd.

Hotpo is short for half or triple-plus-one.

Solution

The logic is "if the input is odd, return 3 times it plus 1, otherwise return half of it". The R structure is an `if-then-else`:

```
hotpo1 = function(x) {
    if (is_odd(x))
        3 * x + 1 else x/2
}
```

In R, the condition that is tested goes in brackets, and then if the value-if-true and the value-if-false are single statements, you just type them. (If they are more complicated than that, you put them in curly brackets.) Now you see the value of writing `is_odd` earlier; this code almost looks like the English-language description of the sequence. If we had not written `is_odd` before, the condition would have looked something like

```
if (x%%2 == 1) 3 * x + 1 else x/2
```

which would have been a lot harder to read.

All right, let's try that out:

```
hotpo1(4)
```

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

```
hotpo1(7)
```

```
## [1] 22
```

```
hotpo1(24)
```

```
## [1] 12
```

That looks all right so far.

(d) Now write a function `hotpo` that will return the whole Collatz sequence for an input x_1. For this, assume that you will eventually get to 1.

Solution

This is a loop, but not a `for` loop (or something that we could do with a `map`), because we don't know how many times we have to go around. This is the kind of thing that we should use a `while` loop for: "keep going while a condition is true". In this case, we should keep going if we haven't reached 1 yet. If we haven't reached 1, we should generate the next value of the sequence and glue it onto what we have so far. To initialize the sequence, we start with the input value. There is an R trick to glue a value onto the sequence, which is to use `c` with a vector and a value, and save it back into the vector:

```
hotpo = function(x) {
    sequence = x
    term = x
    while (term > 1) {
        term = hotpo1(term)
        sequence = c(sequence, term)
    }
    sequence
}
```

I use `term` to hold the current term of the sequence, and overwrite it with the next one (since I don't need the old one any more).

Does it work?

```
hotpo(4)
```

```
## [1] 4 2 1
```

```
hotpo(12)
```

```
##  [1] 12  6  3 10  5 16  8  4  2  1
```

```
hotpo(97)
```

```
##   [1]   97  292  146   73  220  110   55  166
##   [9]   83  250  125  376  188   94   47  142
##  [17]   71  214  107  322  161  484  242  121
##  [25]  364  182   91  274  137  412  206  103
##  [33]  310  155  466  233  700  350  175  526
##  [41]  263  790  395 1186  593 1780  890  445
##  [49] 1336  668  334  167  502  251  754  377
##  [57] 1132  566  283  850  425 1276  638  319
```

```
##  [65]  958  479 1438  719 2158 1079 3238 1619
##  [73] 4858 2429 7288 3644 1822  911 2734 1367
##  [81] 4102 2051 6154 3077 9232 4616 2308 1154
##  [89]  577 1732  866  433 1300  650  325  976
##  [97]  488  244  122   61  184   92   46   23
## [105]   70   35  106   53  160   80   40   20
## [113]   10    5   16    8    4    2    1
```

97 is a wild ride, but it does eventually get to 1.

Extra: where I originally saw this, which was "Metamagical Themas" by Douglas Hofstadter, he was illustrating the programming language Lisp and the process of recursion, whereby you define a function in terms of itself. This one is a natural for that, because the Collatz sequence starting at x is x along with the Collatz sequence starting at the next term. For example, if you start at 12, the next term is 6, so that the Collatz sequence starting at 12 is 12 followed by the Collatz sequence starting at 6. There is no dependence any further back. You can do recursion in R also; there is no problem with a function calling itself:

```
hotpo_rec = function(x) {
    if (x == 1)
        1 else c(x, hotpo_rec(hotpo1(x)))
}
```

Recursive functions have two parts: a "base case" that says how you know you are done (the 1 here), and a "recursion" that says how you move to a simpler case, here working out the next term, getting the whole sequence for that, and gluing the input onto the front. It seems paradoxical that you define a function in terms of itself, but what you are doing is calling a simpler sequence, in this case one that is length one shorter than the sequence for the original input. Thus, we hope, we will eventually reach 1.

Nobody knows whether you *always* get to 1, but also nobody has ever found a case where you don't. Collatz's conjecture, that you will get to 1 eventually, is known to be true for all starting x_1 up to some absurdly large number, but not for *all* starting points.

Does it work?

```
hotpo_rec(12)
```

```
##  [1] 12  6  3 10  5 16  8  4  2  1
```

```
hotpo_rec(97)
```

```
##   [1]   97  292  146   73  220  110   55  166
##   [9]   83  250  125  376  188   94   47  142
##  [17]   71  214  107  322  161  484  242  121
##  [25]  364  182   91  274  137  412  206  103
##  [33]  310  155  466  233  700  350  175  526
```

```
##  [41]  263  790  395 1186  593 1780  890  445
##  [49] 1336  668  334  167  502  251  754  377
##  [57] 1132  566  283  850  425 1276  638  319
##  [65]  958  479 1438  719 2158 1079 3238 1619
##  [73] 4858 2429 7288 3644 1822  911 2734 1367
##  [81] 4102 2051 6154 3077 9232 4616 2308 1154
##  [89]  577 1732  866  433 1300  650  325  976
##  [97]  488  244  122   61  184   92   46   23
## [105]   70   35  106   53  160   80   40   20
## [113]   10    5   16    8    4    2    1
```

It does.

Recursive functions are often simple to understand, but they are not always very efficient. They can take a lot of memory, because they have to handle the intermediate calls to the function, which they have to save to use later (in the case of `hotpo_rec(97)` there are a lot of those). Recursive functions are often paired with a technique called "memoization", where each time you calculate the function's value, you *save* it in another array. The first thing you do in the recursive function is to check whether you already have the answer, in which case you just look it up and return it. It was a lot of work here to calculate the sequence from 97, but if we had saved the results, we would already have the answers for 292, 146, 73, 220 and so on, and getting those later would be a table lookup rather than another recursive calculation.

(e) Write two (very small) functions that take an entire sequence as input and return (i) the length of the sequence and (ii) the maximum value it attains.

Solution

These are both one-liners. Call the input whatever you like:

```
hotpo_len = function(sequence) length(sequence)
hotpo_max = function(sequence) max(sequence)
```

Because they are one-liners, you don't even need the curly brackets, although there's no problem if they are there.

Testing:

```
hotpo_len(hotpo(12))
```

```
## [1] 10
```

```
hotpo_max(hotpo(97))
```

```
## [1] 9232
```

This checks with what we had before.

(f) Make a data frame consisting of the values 11 through 20, and, using `tidyverse` ideas, obtain a data frame containing the Collatz sequences starting at each of those values, along with their lengths and their maximum values. Which sequence is longest? Which one goes up highest?

Solution

This one uses `map` ideas: an actual `map` for the sequence, and `map_dbl` for the length and maximum value, since these are both actually integers but the calculation in our functions actually uses decimals. Thus, a pipeline:

I should have been more careful in my functions to make sure everything was integers, and, in particular, to do integer division by 2 because I knew that this division was going to come out even.

```
tibble(x = 11:20) %>% mutate(sequence = map(x,
    ~hotpo(.))) %>% mutate(length = map_dbl(sequence,
    ~hotpo_len(.))) %>% mutate(high = map_dbl(sequence,
    ~hotpo_max(.)))
```

```
## # A tibble: 10 x 4
##        x sequence   length  high
##    <int> <list>      <dbl> <dbl>
##  1    11 <dbl [15]>     15    52
##  2    12 <dbl [10]>     10    16
##  3    13 <dbl [10]>     10    40
##  4    14 <dbl [18]>     18    52
##  5    15 <dbl [18]>     18   160
##  6    16 <dbl [5]>       5    16
##  7    17 <dbl [13]>     13    52
##  8    18 <dbl [21]>     21    52
##  9    19 <dbl [21]>     21    88
## 10    20 <dbl [8]>       8    20
```

First, we obtain a list-column containing the sequences, then two ordinary columns of their lengths and their maximum values. The `map` is the same each time: for each of the first thing, run the function on *it*, where "it" is represented by the dot.

The sequences for 18 and 19 are the longest, but the sequence for 15 goes up the highest.

16
Logistic regression

```r
library(tidyverse)
```

```
## -- Attaching packages ---- tidyverse 1.2.1 --

## v ggplot2 3.1.0     v purrr   0.2.5
## v tibble  1.4.2     v dplyr   0.7.8
## v tidyr   0.8.1     v stringr 1.3.1
## v readr   1.1.1     v forcats 0.3.0

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

16.1 Finding wolf spiders on the beach

A team of Japanese researchers were investigating what would cause the burrowing wolf spider Lycosa ishikariana* to be found on a beach. They hypothesized that it had to do with the size of the grains of sand on the beach. They went to 28 beaches in Japan, measured the average sand grain size (in millimetres), and observed the presence or absence of this particular spider on each beach. The data are in link.

(a) Why would logistic regression be better than regular regression in this case?

Solution

Because the response variable, whether or not the spider is present, is a categorical yes/no success/failure kind of variable rather than a quantitative numerical one, and when you have this kind of response variable, this is when you want to use logistic regression.

(b) Read in the data and check that you have something sensible. (Look at the data file first: the columns are aligned but the headers are not aligned with them.)

Solution

The nature of the file means that you need `read_table2`:

```
my_url = "http://www.utsc.utoronto.ca/~butler/d29/spiders.txt"
spider = read_table2(my_url)
```

```
## Parsed with column specification:
## cols(
##   Grain.size = col_double(),
##   Spiders = col_character()
## )
```

```
spider
```

```
## # A tibble: 28 x 2
##    Grain.size Spiders
##         <dbl> <chr>
##  1      0.245 absent
##  2      0.247 absent
##  3      0.285 present
##  4      0.299 present
##  5      0.327 present
##  6      0.347 present
##  7      0.356 absent
##  8      0.36  present
##  9      0.363 absent
## 10      0.364 present
## # ... with 18 more rows
```

We have a numerical sand grain size and a categorical variable `Spiders` that indicates whether the spider was present or absent. As we were expecting. (The categorical variable is actually text or `chr`, which will matter in a minute.)

(c) Make a boxplot of sand grain size according to whether the spider is present or absent. Does this suggest that sand grain size has something to do with presence or absence of the spider?

Solution

```
ggplot(spider, aes(x = Spiders, y = Grain.size)) +
    geom_boxplot()
```

The story seems to be that when spiders are present, the sand grain size tends to be larger. So we would expect to find some kind of useful relationship in the logistic regression.

Note that we have reversed the cause and effect here: in the boxplot we are asking "given that the spider is present or absent, how big are the grains of sand?", whereas the logistic regression is asking "given the size of the grains of sand, how likely is the spider to be present?". But if one variable has to do with the other, we would hope to see the link either way around.

(d) Fit a logistic regression predicting the presence or absence of spiders from the grain size, and obtain its `summary`. Note that you will need to do something with the response variable first.

Solution

The presence or absence of spiders is our response. This is actually text at the moment:

```
spider
```

```
## # A tibble: 28 x 2
##    Grain.size Spiders
##         <dbl> <chr>
##  1      0.245 absent
##  2      0.247 absent
##  3      0.285 present
##  4      0.299 present
##  5      0.327 present
##  6      0.347 present
##  7      0.356 absent
##  8      0.36  present
##  9      0.363 absent
```

```
## 10      0.364 present
## # ... with 18 more rows
```

so we need to make a factor version of it first. I'm going to live on the edge and overwrite everything:

```
spider = spider %>% mutate(Spiders = factor(Spiders))
spider
```

```
## # A tibble: 28 x 2
##    Grain.size Spiders
##         <dbl> <fct>
##  1      0.245 absent
##  2      0.247 absent
##  3      0.285 present
##  4      0.299 present
##  5      0.327 present
##  6      0.347 present
##  7      0.356 absent
##  8      0.36  present
##  9      0.363 absent
## 10      0.364 present
## # ... with 18 more rows
```

`Spiders` is now a factor, correctly. (Sometimes a text variable gets treated as a factor, sometimes it needs to be an explicit `factor`. This is one of those times.) Now we go ahead and fit the model. I'm naming this as response-with-a-number, so I still have the Capital Letter. Any choice of name is good, though.

```
Spiders.1 = glm(Spiders ~ Grain.size, family = "binomial",
    data = spider)
summary(Spiders.1)
```

```
##
## Call:
## glm(formula = Spiders ~ Grain.size, family = "binomial", data = spider)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -1.7406  -1.0781   0.4837   0.9809   1.2582
##
## Coefficients:
##             Estimate Std. Error z value
## (Intercept)   -1.648      1.354  -1.217
## Grain.size     5.122      3.006   1.704
```

```
##             Pr(>|z|)
## (Intercept)   0.2237
## Grain.size    0.0884 .
## ---
## Signif. codes:
##   0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1  ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 35.165  on 27  degrees of freedom
## Residual deviance: 30.632  on 26  degrees of freedom
## AIC: 34.632
##
## Number of Fisher Scoring iterations: 5
```

(e) Is there a significant relationship between grain size and presence or absence of spiders at the $\alpha = 0.10$ level? Explain briefly.

Solution

The P-value on the `Grain.size` line is *just* less than 0.10 (it is 0.088), so there is *just* a significant relationship. It isn't a very strong significance, though. This might be because we don't have that much data: even though we have 28 observations, which, on the face of it, is not a very small sample size, each observation doesn't tell us much: only whether the spider was found on that beach or not. Typical sample sizes in logistic regression are in the hundreds — the same as for opinion polls, because you're dealing with the same kind of data. The weak significance here lends some kind of weak support to the researchers' hypothesis, but I'm sure they were hoping for something better.

(f) We want to obtain predicted probabilities of spider presence for grain sizes of 0.2, 0.5, 0.8 and 1.1 millimetres. Do this by creating a new data frame with one column called `Grain.size` (watch the capital letter!) and those four values, and then feed this into `predict`. I only want predicted probabilities, not any kind of intervals.

Solution

Something like this. I like to save all the values first, and *then* make a data frame of them, but you can do it in one go if you prefer:

```
Grain.sizes = c(0.2, 0.5, 0.8, 1.1)
spider.new = tibble(Grain.size = Grain.sizes)
spider.new
```

```
## # A tibble: 4 x 1
##   Grain.size
##        <dbl>
## 1        0.2
## 2        0.5
## 3        0.8
## 4        1.1
```

```
pred.prob = predict(Spiders.1, spider.new, type = "response")
cbind(spider.new, pred.prob)
```

```
##   Grain.size pred.prob
## 1        0.2 0.3490280
## 2        0.5 0.7136446
## 3        0.8 0.9205335
## 4        1.1 0.9817663
```

(Second attempt: on my first, the `tibble` contained `Grain.Sizes`!)

This exemplifies my preferred technique in these things: I store the values to predict for in a vector with a *plural* name. The column names in the data frame I make have to have the same (singular) name as the original data, so the layout in making the data frame is "singular equals plural". You'll see a lot of this as we go through the course.

I think the data frame of values to predict for is called `spider.new` because the original one is called `spider`. I have to stop and think about my naming conventions.

The above is all I need. (The stuff below is my extra comments.)

Note that the probabilities don't go up linearly. (If they did, they would soon get bigger than 1!). It's actually the *log-odds* that go up linearly. In fact, if you leave out the `type` on the `predict`, log-odds is what you get:

```
pred.log.odds = predict(Spiders.1, spider.new)
cbind(spider.new, pred.log.odds)
```

```
##   Grain.size pred.log.odds
## 1        0.2    -0.6233144
## 2        0.5     0.9131514
## 3        0.8     2.4496172
## 4        1.1     3.9860831
```

The meaning of that slope coefficient in the `summary`, which is about 5, is that if you increase grain size by 1, you increase the log-odds by 5. In the table above, we increased the grain size by 0.3 each time, so we would expect to increase the log-odds by $(0.3)(5) = 1.5$, which is (to this accuracy) what happened.

Log-odds are hard to interpret. Odds are a bit easier, and to get them, we have to `exp` the log-odds:

```
cbind(spider.new, odds = exp(pred.log.odds))
```

```
##   Grain.size       odds
## 1        0.2  0.5361644
## 2        0.5  2.4921640
## 3        0.8 11.5839120
## 4        1.1 53.8435749
```

Thus, with each step of 0.3 in grain size, we *multiply* the odds of finding a spider by about

```
exp(1.5)
```

```
## [1] 4.481689
```

or about 4.5 times. If you're a gambler, this might give you a feel for how large the effect of grain size is. Or, of course, you can just look at the probabilities.

(g) Given your previous work in this question, does the trend you see in your predicted probabilities surprise you? Explain briefly.

Solution

My predicted probabilities go up as grain size goes up. This fails to surprise me for a couple of reasons: first, in my boxplots, I saw that grain size tended to be larger when spiders were present, and second, in my logistic regression `summary`, the slope was positive, so likewise spiders should be more likely as grain size goes up. Observing just one of these things is enough, though of course it's nice if you can spot both.

16.2 *Killing aphids*

An experiment was designed to examine how well the insecticide rotenone kills aphids that feed on the chrysanthemum plant called *Macrosiphoniella sanborni*. The explanatory variable is the log concentration (in milligrams per litre) of the insecticide. At each of the five different concentrations, approximately 50 insects were exposed. The number of insects exposed at each concentration, and the number killed, are shown below.

Log-Concentration	Insects exposed	Number killed
0.96	50	6

```
1.33                        48              16
1.63                        46              24
2.04                        49              42
2.32                        50              44
```

(a) Get these data into R. You can do this by copying the data into a file and reading that into R (creating a data frame), or you can enter the data manually into R using `c`, since there are not many values. In the latter case, you can create a data frame or not, as you wish. Demonstrate that you have the right data in R.

Solution

There are a couple of ways. My current favourite is the `tidyverse`-approved `tribble` method. A `tribble` is a "transposed `tibble`", in which you copy and paste the data, inserting column headings and commas in the right places:

```
dead_bugs = tribble(~log_conc, ~exposed, ~killed,
    0.96, 50, 6, 1.33, 48, 16, 1.63, 46, 24, 2.04,
    49, 42, 2.32, 50, 44)
dead_bugs
```

```
## # A tibble: 5 x 3
##   log_conc exposed killed
##      <dbl>   <dbl>  <dbl>
## 1     0.96      50      6
## 2     1.33      48     16
## 3     1.63      46     24
## 4     2.04      49     42
## 5     2.32      50     44
```

Note that the last data value has no comma after it, but instead has the closing bracket of `tribble`. If you are clever in R Studio, you can insert a column of commas all at once (using "multiple cursors"). I used to do it like this. I make vectors of each column using `c` and then glue the columns together into a data frame:

```
log_conc = c(0.96, 1.33, 1.63, 2.04, 2.32)
exposed = c(50, 48, 46, 49, 50)
killed = c(6, 16, 24, 42, 44)
dead_bugs2 = tibble(log_conc, exposed, killed)
dead_bugs2
```

```
## # A tibble: 5 x 3
##   log_conc exposed killed
##      <dbl>   <dbl>  <dbl>
```

```
## 1     0.96      50      6
## 2     1.33      48     16
## 3     1.63      46     24
## 4     2.04      49     42
## 5     2.32      50     44
```

The values are correct — I checked them.

Now you see why `tribble` stands for "transposed tibble": if you want to construct a data frame by hand, you have to work with columns and then glue them together, but `tribble` allows you to work "row-wise" with the data as you would lay it out on the page.

The other obvious way to read the data values without typing them is to copy them into a file and read *that*. The values as laid out are aligned in columns. They might be separated by tabs, but they are not. (It's hard to tell without investigating, though a tab is by default eight spaces and these values look farther apart than that.) I copied them into a file `exposed.txt` in my current folder (or use `file.choose`):

```
bugs2 = read_table("exposed.txt")
```

```
## Parsed with column specification:
## cols(
##   `Log-Concentration` = col_double(),
##   `Insects exposed` = col_integer(),
##   Number = col_integer(),
##   killed = col_character()
## )
```

```
bugs2
```

```
## # A tibble: 5 x 4
##   `Log-Concentrat~ `Insects expose~ Number
##              <dbl>            <int>  <int>
## 1             0.96               50      6
## 2             1.33               48     16
## 3             1.63               46     24
## 4             2.04               49     42
## 5             2.32               50     44
## # ... with 1 more variable: killed <chr>
```

This didn't quite work: the last column `Number killed` got split into two, with the actual number killed landing up in `Number` and the column `killed` being empty. If you look at the data file, the data values for `Number killed` are actually aligned with the word `Number`, which is why it came out this way. Also, you'll note, the column

names have those "backticks" around them, because they contain illegal characters like a minus sign and spaces. Perhaps a good way to pre-empt all these problems is to make a copy of the data file with the illegal characters replaced by underscores, which is my file `exposed2.txt`:

My daughter learned the word pre-empt because we like to play a bridge app on my phone; in the game of bridge, you make a pre-emptive bid when you have no great strength but a lot of cards of one suit, say seven, and it won't be too bad if that suit is trumps, no matter what your partner has. If you have a weak hand with a lot of cards in one suit, your opponents are probably going to be able to bid and make something, so you pre-emptively bid first to try and make it difficult for them.

```
bugs2 = read_table("exposed2.txt")
```

```
## Parsed with column specification:
## cols(
##   Log_Concentration = col_double(),
##   Insects_exposed = col_integer(),
##   Number_killed = col_integer()
## )
```

```
bugs2
```

```
## # A tibble: 5 x 3
##   Log_Concentrati~ Insects_exposed
##              <dbl>           <int>
## 1             0.96              50
## 2             1.33              48
## 3             1.63              46
## 4             2.04              49
## 5             2.32              50
## # ... with 1 more variable:
## #   Number_killed <int>
```

This is definitely good. We'd have to be careful with Capital Letters this way, but it's definitely good.

You may have thought that this was a lot of fuss to make about reading in data, but the point is that data can come your way in lots of different forms, and you need to be able to handle whatever you receive so that you can do some analysis with it.

(b)* Looking at the data, would you expect there to be a significant effect of log-concentration? Explain briefly.

Solution

The numbers of insects killed goes up *sharply* as the concentration increases, while the numbers of insects exposed don't change much. So I would expect to see a strong, positive effect of concentration, and I would expect it to be strongly significant, especially since we have almost 250 insects altogether.

(c) We are going to do a logistic regression to predict how likely an insect is to be killed, as it depends on the log-concentration. Create a suitable response variable, bearing in mind (i) that we have lots

of insects exposed to each different concentration, and (ii) what needs to go into each column of the response.

Solution

There needs to be a two-column response variable. The first column needs to be the number of "successes" (insects killed, here) and the second needs to be the number of "failures" (insects that survived). We don't actually have the latter, but we know how many insects were exposed in total to each dose, so we can work it out. Like this:

```
response = dead_bugs %>% mutate(survived = exposed -
    killed) %>% select(killed, survived) %>% as.matrix()
response
```

```
##      killed survived
## [1,]      6       44
## [2,]     16       32
## [3,]     24       22
## [4,]     42        7
## [5,]     44        6
```

`glm` requires an R `matrix` rather than a data frame, so the last stage of our pipeline is to create one (using the same numbers that are in the data frame: all the `as.` functions do is to change what type of thing it is, without changing its contents).

It's also equally good to create the response *outside* of the data frame and use `cbind` to glue its columns together:

```
resp2 = with(dead_bugs, cbind(killed, survived = exposed -
    killed))
resp2
```

```
##      killed survived
## [1,]      6       44
## [2,]     16       32
## [3,]     24       22
## [4,]     42        7
## [5,]     44        6
```

(d) Run a suitable logistic regression, and obtain a summary of the results.

Solution

I think you know how to do this by now:

```
bugs.1 = glm(response ~ log_conc, family = "binomial",
    data = dead_bugs)
summary(bugs.1)
```

```
##
## Call:
## glm(formula = response ~ log_conc, family = "binomial", data = dead_bugs)
##
## Deviance Residuals:
##       1        2        3        4        5
## -0.1963   0.2099  -0.2978   0.8726  -0.7222
##
## Coefficients:
##             Estimate Std. Error z value
## (Intercept)  -4.8923     0.6426  -7.613
## log_conc      3.1088     0.3879   8.015
##             Pr(>|z|)
## (Intercept) 2.67e-14 ***
## log_conc    1.11e-15 ***
## ---
## Signif. codes:
##   0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1  ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 96.6881  on 4  degrees of freedom
## Residual deviance:  1.4542  on 3  degrees of freedom
## AIC: 24.675
##
## Number of Fisher Scoring iterations: 4
```

(e) Does your analysis support your answer to (here)? Explain briefly.

Solution

That's a *very* small P-value, 1.1×10^{-15}, on `log_conc`, so there is no doubt that concentration has an effect on an insect's chances of being killed. This is exactly what I guessed in (here), which I did before looking at the results, honest!

(f) Obtain predicted probabilities of an insect's being killed at each of the log-concentrations in the data set. (This is easier than it sometimes is, because here you don't create a new data frame for `predict`.)

Solution

Just this (notice there are only *two* things going into `predict`):

```
prob = predict(bugs.1, type = "response")
cbind(dead_bugs$log_conc, prob)
```

```
##             prob
## 1 0.96 0.1292158
## 2 1.33 0.3191540
## 3 1.63 0.5436313
## 4 2.04 0.8099321
## 5 2.32 0.9105221
```

or, if you frame everything in terms of the `tidyverse`, turn the predictions from a `matrix` into a `tibble` first, and then use `bind_cols` to glue them together:

```
as_tibble(prob) %>% bind_cols(dead_bugs)
```

```
## # A tibble: 5 x 4
##   value log_conc exposed killed
##   <dbl>    <dbl>   <dbl>  <dbl>
## 1 0.129     0.96      50      6
## 2 0.319     1.33      48     16
## 3 0.544     1.63      46     24
## 4 0.810     2.04      49     42
## 5 0.911     2.32      50     44
```

The advantage of showing the whole input data frame is that you can compare the observed with the predicted. For example, 44 out of 50 insects were killed at log-dose 2.32, which is a proportion of 0.88, pretty close to the prediction of 0.91.

(g) People in this kind of work are often interested in the "median lethal dose". In this case, that would be the log-concentration of the insecticide that kills half the insects. Based on your predictions, roughly what do you think the median lethal dose is?

Solution

The log-concentration of 1.63 is predicted to kill just over half the insects, so the median lethal dose should be a bit less than 1.63. It should not be as small as 1.33, though, since that log-concentration only kills less than a third of the insects. So I would guess somewhere a bit bigger than 1.5. Any guess somewhere in this ballpark is fine: you really cannot be very accurate. This is kind of a strange prediction problem, because we know what the *response* variable should be, and we want to know what the explanatory variable's value is. Normally we do predictions the other way around. So the only way to get a more accurate figure is to try some different log-concentrations, and see which one gets closest to a probability 0.5 of killing the insect. Something like this would work:

```
lc.new = tibble(log_conc = seq(1.5, 1.63, 0.01))
prob = predict(bugs.1, lc.new, type = "response")
cbind(lc.new, prob)
```

```
##    log_conc      prob
## 1      1.50 0.4429568
## 2      1.51 0.4506406
## 3      1.52 0.4583480
## 4      1.53 0.4660754
## 5      1.54 0.4738191
## 6      1.55 0.4815754
## 7      1.56 0.4893406
## 8      1.57 0.4971110
## 9      1.58 0.5048827
## 10     1.59 0.5126521
## 11     1.60 0.5204154
## 12     1.61 0.5281689
## 13     1.62 0.5359087
## 14     1.63 0.5436313
```

The closest one of these to a probability of 0.5 is 0.4971, which goes with a log-concentration of 1.57: indeed, a bit bigger than 1.5 and a bit less than 1.63. The `seq` in the construction of the new data frame is "fill sequence": go from 1.5 to 1.63 in steps of 0.01. The rest of it is the same as before.

Now, of course this is only our "best guess", like a single-number prediction in regression. There is uncertainty attached to it (because the actual logistic regression might be different from the one we estimated), so we ought to provide a confidence interval for it. But I'm riding the bus as I type this, so I can't look it up right now.

16.3 *The effects of Substance A*

In a dose-response experiment, animals (or cell cultures or human subjects) are exposed to some toxic substance, and we observe how many of them show some sort of response. In this experiment, a mysterious Substance A is exposed at various doses to 100 cells at each dose, and the number of cells at each dose that suffer damage is recorded. The doses were 10, 20, ... 70 (mg), and the number of damaged cells out of 100 were respectively 10, 28, 53, 77, 91, 98, 99.

(a) Find a way to get these data into R, and show that you have managed to do so successfully.

Solution

There's not much data here, so we don't need to create a file, although you can do so if you like (in the obvious way: type the doses and damaged cell numbers into a `.txt` file or spreadsheet and read in with the appropriate `read_` function). Or, use a `tribble`:

```
dr = tribble(~dose, ~damaged, 10, 10, 20, 28,
    30, 53, 40, 77, 50, 91, 60, 98, 70, 99)
dr
```

```
## # A tibble: 7 x 2
##    dose damaged
##   <dbl>   <dbl>
## 1    10      10
## 2    20      28
## 3    30      53
## 4    40      77
## 5    50      91
## 6    60      98
## 7    70      99
```

Or, make a data frame with the values typed in:

```
dr2 = tibble(dose = seq(10, 70, 10), damaged = c(10,
    28, 53, 77, 91, 98, 99))
dr2
```

```
## # A tibble: 7 x 2
##    dose damaged
##   <dbl>   <dbl>
## 1    10      10
## 2    20      28
## 3    30      53
## 4    40      77
## 5    50      91
## 6    60      98
## 7    70      99
```

`seq` fills a sequence "10 to 70 in steps of 10", or you can just list the doses.

I like this better than making two columns *not* attached to a data frame, but that will work as well.

For these, find a way you like, and stick with it.

(b) Would you expect to see a significant effect of dose for these data? Explain briefly.

Solution

The number of damaged cells goes up sharply as the dose goes up (from a very small number to almost all of them). So I'd expect to see a strongly significant effect of dose. This is far from something that would happen by chance.

(c) Fit a logistic regression modelling the probability of a cell being damaged as it depends on dose. Display the results. (Comment on them is coming later.)

Solution

This has a bunch of observations at each dose (100 cells, in fact), so we need to do the two-column response thing: the successes in the first column and the failures in the second. This means that we first need to calculate the number of cells at each dose that were not damaged, by subtracting the number that *were* damaged from 100. R makes this easier than you'd think, as you see. A `mutate` is the way to go: create a new column in `dr` and save back in `dr` (because I like living on the edge):

```
dr = dr %>% mutate(undamaged = 100 - damaged)
dr
```

```
## # A tibble: 7 x 3
##    dose damaged undamaged
##   <dbl>   <dbl>     <dbl>
## 1    10      10        90
## 2    20      28        72
## 3    30      53        47
## 4    40      77        23
## 5    50      91         9
## 6    60      98         2
## 7    70      99         1
```

The programmer in you is probably complaining "but, 100 is a number and `damaged` is a vector of 7 numbers. How does R know to subtract *each one* from 100?" Well, R has what's known as "recycling rules": if you try to add or subtract (or elementwise multiply or divide) two vectors of different lengths, it recycles the smaller one by repeating it until it's as long as the longer one. So rather than `100-damaged` giving an error, it does what you want.

The usual application of this is to combine a number with a vector. If you try to subtract a length-2 vector from a length-6 vector, R will repeat the shorter one three times and do the subtraction, but if you try to subtract a length-2 vector from a length-*7* vector, where you'd have to repeat the shorter one a fractional number of times, R will do it, but you'll get a warning, because this is probably *not* what you wanted. Try it and see.

I took the risk of saving the new data frame over the old one. If it had failed for some reason, I could have started again.

Now we have to make our response "matrix" with two columns, using `cbind`:

```
response = with(dr, cbind(damaged, undamaged))
response
```

```
##      damaged undamaged
## [1,]      10        90
## [2,]      28        72
## [3,]      53        47
## [4,]      77        23
## [5,]      91         9
## [6,]      98         2
## [7,]      99         1
```

Note that each row adds up to 100, since there were 100 cells at each dose. This is actually trickier than it looks: the two things in `cbind` are columns (vectors), and `cbind` glues them together to make an R `matrix`:

```
class(response)
```

```
## [1] "matrix"
```

which is what `glm` needs here, even though it looks a lot like a data frame (which wouldn't work here). This *would* be a data frame:

```
dr %>% select(damaged, undamaged) %>% class()
```

```
## [1] "tbl_df"     "tbl"        "data.frame"
```

and would therefore be the wrong thing to give `glm`. So I had to do it with `cbind`, or use some other trickery, like this:

```
resp = dr %>% select(damaged, undamaged) %>% as.matrix()
class(resp)
```

```
## [1] "matrix"
```

Now we fit our model:

```
cells.1 = glm(response ~ dose, family = "binomial",
    data = dr)
summary(cells.1)
```

```
##
## Call:
## glm(formula = response ~ dose, family = "binomial", data = dr)
##
## Deviance Residuals:
##        1         2         3         4
## -0.16650   0.28794  -0.02092  -0.20637
##        5         6         7
## -0.21853   0.54693  -0.06122
```

```
##
## Coefficients:
##              Estimate Std. Error z value
## (Intercept) -3.275364   0.278479  -11.76
## dose         0.113323   0.008315   13.63
##             Pr(>|z|)
## (Intercept)   <2e-16 ***
## dose          <2e-16 ***
## ---
## Signif. codes:
##   0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1  ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 384.13349  on 6  degrees of freedom
## Residual deviance:   0.50428  on 5  degrees of freedom
## AIC: 31.725
##
## Number of Fisher Scoring iterations: 4
```

(d) Does your output indicate that the probability of damage really does *increase* with dose? (There are two things here: is there really an effect, and which way does it go?)

Solution

The slope of `dose` is significantly nonzero (P-value less than 2.2×10^{-16}, which is as small as it can be). The slope itself is *positive*, which means that as dose goes up, the probability of damage goes up. That's all I needed, but if you want to press on: the slope is 0.113, so an increase of 1 in dose goes with an increase of 0.113 in the *log-odds* of damage. Or it multiplies the odds of damage by $\exp(0.113)$. Since 0.113 is small, this is about 1.113 (mathematically, $e^x \simeq 1 + x$ if x is small), so that the increase is about 11%. If you like, you can get a rough CI for the slope by going up and down twice its standard error (this is the usual approximately-normal thing). Here that would be

```r
0.113323 + c(-2, 2) * 0.008315
```

```
## [1] 0.096693 0.129953
```

I thought about doing that in my head, but thought better of it, since I have R just sitting here. The interval is short (we have lots of data) and definitely does not contain zero.

(e) Obtain predicted damage probabilities for the doses that were actually used in the experiment. (This is easier than some of our

other predictions: if you are using the original data, you only need to feed in the fitted model object and not a data frame of new data.) Display your predicted probabilities next to the doses and the observed numbers of damaged cells.

Solution

The remaining care that this needs is to make sure that we do indeed get predicted probabilities, and then some `cbind`ing to get the right kind of display:

```
p = predict(cells.1, type = "response")
with(dr, cbind(dose, damaged, p))
```

```
##   dose damaged         p
## 1   10      10 0.1050689
## 2   20      28 0.2671957
## 3   30      53 0.5310440
## 4   40      77 0.7786074
## 5   50      91 0.9161232
## 6   60      98 0.9713640
## 7   70      99 0.9905969
```

If you forget the `type="response"` you'll get predicted log-odds, which are not nearly so easy to interpret.

(f) Looking at the predicted probabilities, would you say that the model fits well? Explain briefly.

Solution

There were 100 cells tested at each dose, so that the predicted probabilities should be close to the observed numbers divided by 100. They are in fact *very* close to this, so the model fits very well. Your actual words are a judgement call, so precisely what you say doesn't matter so much, but *I* think this model fit is actually closer than you could even hope to expect, it's that good. But, your call. I think your answer ought to contain "close" or "fits well" at the very least.

16.4 *What makes an animal get infected?*

Some animals got infected with a parasite. We are interested in whether the likelihood of infection depends on any of the age, weight and sex of the animals. The data are at link. The values are separated by tabs.

(a) Read in the data and take a look at the first few lines. Is this one animal per line, or are several animals with the same age, weight

and sex (and infection status) combined onto one line? How can you tell?

Solution

The usual beginnings, bearing in mind the data layout:

```
my_url = "http://www.utsc.utoronto.ca/~butler/d29/infection.txt"
infect = read_tsv(my_url)
```

```
## Parsed with column specification:
## cols(
##   infected = col_character(),
##   age = col_integer(),
##   weight = col_integer(),
##   sex = col_character()
## )
```

```
infect
```

```
## # A tibble: 81 x 4
##    infected   age weight sex
##    <chr>    <int>  <int> <chr>
##  1 absent       2      1 female
##  2 absent       9     13 female
##  3 present     15      2 female
##  4 absent      15     16 female
##  5 absent      18      2 female
##  6 absent      20      9 female
##  7 absent      26     13 female
##  8 present     42      6 female
##  9 absent      51      9 female
## 10 present     52      6 female
## # ... with 71 more rows
```

Success. This appears to be one animal per line, since there is no indication of frequency (of "how many"). If you were working as a consultant with somebody's data, this would be a good thing to confirm with them before you went any further.

You can check a few more lines to convince yourself:

```
infect %>% print(n = 20)
```

```
## # A tibble: 81 x 4
##    infected   age weight sex
##    <chr>    <int>  <int> <chr>
##  1 absent       2      1 female
##  2 absent       9     13 female
```

```
##  3 present     15      2 female
##  4 absent      15     16 female
##  5 absent      18      2 female
##  6 absent      20      9 female
##  7 absent      26     13 female
##  8 present     42      6 female
##  9 absent      51      9 female
## 10 present     52      6 female
## 11 present     59     12 female
## 12 absent      68     10 female
## 13 absent      72     15 female
## 14 present     73      1 female
## 15 absent      78     15 female
## 16 absent      80     16 female
## 17 present     82     14 female
## 18 present     91     12 female
## 19 present    105      5 female
## 20 present    114      8 female
## # ... with 61 more rows
```

and the story is much the same. The other hint is that you have actual categories of response, which usually indicates one individual per row, but not always. If it doesn't, you have some extra work to do to bash it into the right format.

Extra: let's see whether we can come up with an example of that. I'll make a smaller example, and perhaps the place to start is "all possible combinations" of a few things. If you haven't seen `crossing` before, skip ahead to part (here):

```
d = crossing(age = c(10, 12), gender = c("f",
    "m"), infected = c("y", "n"))
d
```

```
## # A tibble: 8 x 3
##     age gender infected
##   <dbl> <chr>  <chr>
## 1    10 f      n
## 2    10 f      y
## 3    10 m      n
## 4    10 m      y
## 5    12 f      n
## 6    12 f      y
## 7    12 m      n
## 8    12 m      y
```

These might be one individual per row, or they might be more

than one, as they would be if we have a column of frequencies:

```
d = d %>% mutate(freq = c(12, 19, 17, 11, 18,
    26, 16, 8))
d
```

```
## # A tibble: 8 x 4
##     age gender infected  freq
##   <dbl> <chr>  <chr>    <dbl>
## 1    10 f      n           12
## 2    10 f      y           19
## 3    10 m      n           17
## 4    10 m      y           11
## 5    12 f      n           18
## 6    12 f      y           26
## 7    12 m      n           16
## 8    12 m      y            8
```

Now, these are multiple observations per row (the presence of frequencies means there's no doubt about that), but the format is wrong: `infected` is my response variable, and we want the frequencies of `infected` being `y` or `n` in *separate* columns — that is, we have to *untidy* the data a bit to make it suitable for modelling. This is `spread`, the opposite of `gather`:

```
d %>% spread(infected, freq)
```

```
## # A tibble: 4 x 4
##     age gender     n     y
##   <dbl> <chr>  <dbl> <dbl>
## 1    10 f         12    19
## 2    10 m         17    11
## 3    12 f         18    26
## 4    12 m         16     8
```

Now you can pull out the columns `y` and `n` and make them into your response, and predict that from `age` and `gender`.

The moral of this story is that if you are going to have multiple observations per row, you probably want the combinations of *explanatory* variables one per row, but you want the categories of the *response* variable in separate columns.

Back to where we were the rest of the way.

(b)* Make suitable plots or summaries of `infected` against each of the other variables. (You'll have to think about `sex`, um, you'll have to think about the `sex` variable, because it too is categorical.) Anything sensible is OK here. You might like to think back to what we did in

Question~here for inspiration. (You can also investigate `table`, which does cross-tabulations.)

Solution

What comes to my mind for the numerical variables `age` and `weight` is boxplots:

```
ggplot(infect, aes(x = infected, y = age)) + geom_boxplot()
```

```
ggplot(infect, aes(x = infected, y = weight)) +
    geom_boxplot()
```

The variables `sex` and `infected` are both categorical. I guess a good plot for those would be some kind of grouped bar plot, which I have to think about. So let's first try a numerical summary, a cross-tabulation, which is gotten via `table`:

```
with(infect, table(sex, infected))
```

```
##         infected
## sex      absent present
##   female     17      11
##   male       47       6
```

Or, if you like the `tidyverse`:

```
infect %>% count(sex, infected)
```

```
## # A tibble: 4 x 3
##   sex    infected     n
##   <chr>  <chr>    <int>
## 1 female absent      17
## 2 female present     11
## 3 male   absent      47
## 4 male   present      6
```

Now, bar plots. Let's start with one variable. The basic bar plot has categories of a categorical variable along the x-axis and each bar is a count of how many observations were in that category. What is nice about `geom_bar` is that it will do the counting for you, so that the plot is just this:

```
ggplot(infect, aes(x = sex)) + geom_bar()
```

There are about twice as many males as females.

You may think that this looks like a histogram, which it almost does, but there is an important difference: the kind of variable on the x-axis. Here, it is a categorical variable, and you count how many observations fall in each category (at least, `ggplot` does). On a histogram, the x-axis variable is a continuous *numerical* one, like height or weight, and you have to *chop it up* into intervals (and then you count how many observations are in each chopped-up interval).

Technically, on a bar plot, the bars have a little gap between them (as here), whereas the histogram bars are right next to each other, because the right side of one histogram bar is the left side of the next.

All right, two categorical variables. The idea is that you have each bar divided into sub-bars based on the frequencies of a second variable, which is specified by `fill`. Here's the basic idea:

```
ggplot(infect, aes(x = sex, fill = infected)) +
    geom_bar()
```

This is known in the business as a "stacked bar chart". The issue is how much of each bar is blue, which is unnecessarily hard to judge because the male bar is taller. (Here, it is not so bad, because the amount of blue in the male bar is smaller and the bar is also taller. But we got lucky here.)

There are two ways to improve this. One is known as a "grouped bar chart", which goes like this:

```
ggplot(infect, aes(x = sex, fill = infected)) +
    geom_bar(position = "dodge")
```

The absent and present frequencies for females are next to each other, and the same for males, and you can read off how big they are from the y-scale. This is my preferred graph for two (or more than two) categorical variables.

You could switch the roles of `sex` and `infected` and get a different chart, but one that conveys the same information. Try it. (The reason for doing it the way around I did is that being infected or not is the response and `sex` is explanatory, so that on my plot you can ask "out of the males, how many were infected?", which is the way around that makes sense.)

The second way is to go back to stacked bars, but make them the same height, so you can compare the fractions of the bars that are each colour. This is `position="fill"`:

```
ggplot(infect, aes(x = sex, fill = infected)) +
    geom_bar(position = "fill")
```

This also shows that more of the females were infected than the males, but without getting sidetracked into the issue that there were more males to begin with.

I wrote this question in early 2017. At that time, I wrote (quote):

I learned about this one approximately two hours ago. I just ordered Hadley Wickham's new book "R for Data Science" from Amazon, and it arrived today. It's in there. (A good read, by the way. I'm thinking of using it as a recommended text next year.) As is so often the way with `ggplot`, the final answer looks very simple, but there is a lot of thinking required to get there, and you end up having even more respect for Hadley Wickham for the clarity of thinking that enabled this to be specified in a simple way.

(end quote)

(c) Which, if any, of your explanatory variables appear to be related to `infected`? Explain briefly.

Solution

Let's go through our output from (here). In terms of `age`, when infection is present, animals are (slightly) older. So there might be a small age effect. Next, when infection is present, weight is typically a *lot* less. So there ought to be a big weight effect. Finally, from the table, females are somewhere around 50-50 infected or not, but very few males are infected. So there ought to be a big `sex` effect as well. This also appears in the grouped bar plot, where the red ("absent") bar for males is much taller than the blue ("present") bar, but for females the two bars are almost the same height. So the story is that we would expect a significant effect of `sex` and `weight`, and maybe of `age` as well.

(d) Fit a logistic regression predicting `infected` from the other three variables. Display the `summary`.

Solution

Thus:

```
infect.1 = glm(infected ~ age + weight + sex,
    family = "binomial", data = infect)
```

```
## Error in eval(family$initialize): y values must be 0 <= y <= 1
```

Oh, I forgot to turn `infected` into a factor. This is the shortcut way to do that:

```
infect.1 = glm(factor(infected) ~ age + weight +
    sex, family = "binomial", data = infect)
summary(infect.1)
```

```
##
## Call:
## glm(formula = factor(infected) ~ age + weight + sex, family = "binomial",
##     data = infect)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -1.9481  -0.5284  -0.3120  -0.1437   2.2525
##
## Coefficients:
##              Estimate Std. Error z value
## (Intercept)  0.609369   0.803288   0.759
## age          0.012653   0.006772   1.868
## weight      -0.227912   0.068599  -3.322
## sexmale     -1.543444   0.685681  -2.251
##             Pr(>|z|)
## (Intercept) 0.448096
## age         0.061701 .
## weight      0.000893 ***
## sexmale     0.024388 *
## ---
## Signif. codes:
##   0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1  ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 83.234  on 80  degrees of freedom
## Residual deviance: 59.859  on 77  degrees of freedom
## AIC: 67.859
##
## Number of Fisher Scoring iterations: 5
```

The "proper" way to do it is to create a new column in the data frame containing the factor version of `infected`. Pipeline fans among you can do it this way:

```
infect.1a <- infect %>% mutate(infected = factor(infected)) %>%
    glm(infected ~ age + weight + sex, family = "binomial",
        data = .)
summary(infect.1a)
```

```
##
## Call:
## glm(formula = infected ~ age + weight + sex, family = "binomial",
##     data = .)
```

```
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -1.9481  -0.5284  -0.3120  -0.1437   2.2525
##
## Coefficients:
##              Estimate Std. Error z value
## (Intercept)  0.609369   0.803288   0.759
## age          0.012653   0.006772   1.868
## weight      -0.227912   0.068599  -3.322
## sexmale     -1.543444   0.685681  -2.251
##             Pr(>|z|)
## (Intercept) 0.448096
## age         0.061701 .
## weight      0.000893 ***
## sexmale     0.024388 *
## ---
## Signif. codes:
##   0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1  ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 83.234  on 80  degrees of freedom
## Residual deviance: 59.859  on 77  degrees of freedom
## AIC: 67.859
##
## Number of Fisher Scoring iterations: 5
```

Either way is good, and gives the same answer. The second way uses the `data=.` trick to ensure that the input data frame to `glm` is the output from the previous step, the one with the factor version of `infected` in it. The `data=.` is needed because `glm` requires a model formula first rather than a data frame (if the data were first, you could just omit it).

(e)* Which variables, if any, would you consider removing from the model? Explain briefly.

Solution

This is the same idea as in multiple regression: look at the end of the line for each variable to get its individual P-value, and if that's not small, you can take that variable out. `age` has a P-value of 0.062, which is (just) larger than 0.05, so we can consider removing this variable. The other two P-values, 0.00089 and 0.024, are definitely less than 0.05, so those variables should stay. Alternatively, you can say that the P-value for `age` is small enough to be interesting, and

therefore that age should stay. That's fine, but then you need to be consistent in the next part.

(f) Are the conclusions you drew in (here) and (here) consistent, or not? Explain briefly.

Solution

I think they are extremely consistent. When we looked at the plots, we said that weight and sex had large effects, and they came out definitely significant. There was a small difference in age between the infected and non-infected groups, and age came out borderline significant (with a P-value definitely larger than for the other variables, so that the evidence of its usefulness was weaker).

(g)* The first and third quartiles of age are 26 and 130; the first and third quartiles of weight are 9 and 16. Obtain predicted probabilities for all combinations of these and sex. (You'll need to start by making a new data frame, using crossing to get all the combinations.)

Solution

Here's how crossing goes. I'll do it in steps. Note my use of plural names to denote the things I want all combinations of:

```
ages = c(26, 130)
weights = c(9, 16)
sexes = c("female", "male")
infect.new = crossing(age = ages, weight = weights,
    sex = sexes)
infect.new
```

```
## # A tibble: 8 x 3
##     age weight sex
##   <dbl>  <dbl> <chr>
## 1    26      9 female
## 2    26      9 male
## 3    26     16 female
## 4    26     16 male
## 5   130      9 female
## 6   130      9 male
## 7   130     16 female
## 8   130     16 male
```

I could have asked you to include some more values of age and weight, for example the median as well, to get a clearer picture. But that would have made infect.new bigger, so I stopped here.

crossing *makes* a data frame from input vectors, so it doesn't matter if those are different lengths. In fact, it's also possible to make this data frame from things like quartiles stored in a data frame. To

do that (as we did in the hospital satisfaction question), you wrap the whole `crossing` in a `with`.

Next, the predictions:

```
pred = predict(infect.1, infect.new, type = "response")
cbind(infect.new, pred)
```

```
##   age weight    sex       pred
## 1  26      9 female 0.24733693
## 2  26      9   male 0.06560117
## 3  26     16 female 0.06248811
## 4  26     16   male 0.01404012
## 5 130      9 female 0.55058652
## 6 130      9   male 0.20744380
## 7 130     16 female 0.19903348
## 8 130     16   male 0.05041244
```

I didn't ask you to comment on these, since the question is long enough already. But that's not going to stop me!

These are predicted probabilities of infection. (When you have one observation per line, the predictions are of the *second* of the two levels of the response variable. When you make that two-column response, the predictions are of the probability of being in the *first* column. That's what it is. As the young people say, don't @ me.)

The way I remember the one-column-response thing is that the first level is the baseline (as it is in a regression with a categorical explanatory variable), and the second level is the one whose probability is modelled (in the same way that the second, third etc.

levels of a categorical explanatory variable are the ones that appear in the `summary` table).

Let's start with `sex`. The probabilities of a female being infected are all much higher than of a corresponding male (with the same age and weight) being infected. Compare, for example, lines 1 and 2. Or 3 and 4. Etc. So `sex` has a big effect.

What about `weight`? As weight goes from 9 to 16, with everything else the same, the predicted probability of infection goes sharply *down*. This is what we saw before: precisely, the boxplot showed us that infected animals were likely to be less heavy.

Last, `age`. As age goes up, the probabilities go (somewhat) up as well. Compare, for example, lines 1 and 5 or lines 4 and 8. I think this is a less dramatic change than for the other variables, but that's a judgement call.

I got this example from (horrible URL warning) here: link It starts on page 275 in my edition. He goes at the analysis a different way, but he finishes with another issue that I want to show you.

Let's work out the residuals and plot them against our quantitative explanatory variables. I think the best way to do this is `augment` from `broom`, to create a data frame containing the residuals alongside the original data:

```
library(broom)
infect.1a = infect.1 %>% augment()
infect.1a %>% as_tibble()
```

```
## # A tibble: 81 x 11
##    factor.infected.   age weight sex
##  * <fct>            <int>  <int> <chr>
##  1 absent               2      1 fema~
##  2 absent               9     13 fema~
##  3 present             15      2 fema~
##  4 absent              15     16 fema~
##  5 absent              18      2 fema~
##  6 absent              20      9 fema~
##  7 absent              26     13 fema~
##  8 present             42      6 fema~
##  9 absent              51      9 fema~
## 10 present             52      6 fema~
## # ... with 71 more rows, and 7 more
## #   variables: .fitted <dbl>, .se.fit <dbl>,
## #   .resid <dbl>, .hat <dbl>, .sigma <dbl>,
## #   .cooksd <dbl>, .std.resid <dbl>
```

```
ggplot(infect.1a, aes(x = weight, y = .resid)) +
    geom_point() + geom_smooth()
```

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

`infect.1a` is, I think, a genuine `data.frame` rather than a `tibble`.

I don't quite know what to make of that plot. It doesn't look quite random, and yet there are just some groups of points rather than any real kind of trend.

The corresponding plot with age goes the same way:

```
ggplot(infect.1a, aes(x = age, y = .resid)) +
    geom_point() + geom_smooth()
```

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

Crawley found the slightest suggestion of an up-and-down curve in there. I'm not sure I agree, but that's what he saw. As with a regular regression, the residuals against anything should look random, with no trends. (Though the residuals from a logistic regression can be kind of odd, because the response variable can only be 1 or 0.) Crawley tries adding squared terms to the logistic regression, which goes like this. The `glm` statement is long, as they usually are, so it's much easier to use `update`:

```
infect.2 = update(infect.1, . ~ . + I(age^2) +
    I(weight^2))
summary(infect.2)
```

```
##
## Call:
## glm(formula = factor(infected) ~ age + weight + sex + I(age^2) +
##     I(weight^2), family = "binomial", data = infect)
##
## Deviance Residuals:
##      Min        1Q    Median        3Q
## -1.70226  -0.44412  -0.19584  -0.02505
```

```
##      Max
##  2.36653
##
## Coefficients:
##               Estimate Std. Error z value
## (Intercept) -3.4475839  1.7978359  -1.918
## age          0.0829364  0.0360205   2.302
## weight       0.4466284  0.3372352   1.324
## sexmale     -1.2203683  0.7683288  -1.588
## I(age^2)    -0.0004009  0.0002004  -2.000
## I(weight^2) -0.0415128  0.0209677  -1.980
##             Pr(>|z|)
## (Intercept)   0.0552 .
## age           0.0213 *
## weight        0.1854
## sexmale       0.1122
## I(age^2)      0.0455 *
## I(weight^2)   0.0477 *
## ---
## Signif. codes:
##   0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1  ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 83.234  on 80  degrees of freedom
## Residual deviance: 48.620  on 75  degrees of freedom
## AIC: 60.62
##
## Number of Fisher Scoring iterations: 6
```

The squared terms are both *just* significant. The linear terms, `age` and `weight`, have to stay, regardless of their significance. What do the squared terms do to the predictions? Before, there was a clear one-directional trend in the relationships with `age` and `weight`. Has that changed? Let's see. We'll need a few more ages and weights to investigate with. Below, I use `seq` a different way to get a filled series of a desired length:

When you have higher-order terms, you have to keep the lower-order ones as well: higher powers, or interactions (as we see in ANOVA later).

```
weights = seq(9, 16, length.out = 6)
weights
```

```
## [1]  9.0 10.4 11.8 13.2 14.6 16.0
```

```
ages = seq(26, 130, length.out = 6)
ages
```

```
## [1]  26.0  46.8  67.6  88.4 109.2 130.0
```

```
infect.new = crossing(weight = weights, age = ages,
    sex = sexes)
infect.new
```

```
## # A tibble: 72 x 3
##    weight   age sex
##     <dbl> <dbl> <chr>
##  1      9  26   female
##  2      9  26   male
##  3      9  46.8 female
##  4      9  46.8 male
##  5      9  67.6 female
##  6      9  67.6 male
##  7      9  88.4 female
##  8      9  88.4 male
##  9      9 109.  female
## 10      9 109.  male
## # ... with 62 more rows
```

The values are kind of dopey, but they are equally spaced between the two endpoints.

All right, predictions. I'm re-doing the predictions from the previous model (without the squared terms), for annoying technical reasons. Then the data frame pp below will contain the predictions from both models so that we can compare them:

```
pred = predict(infect.2, infect.new, type = "response")
pred.old = predict(infect.1, infect.new, type = "response")
pp = cbind(infect.new, pred.old, pred)
```

This is rather a big data frame, so we'll pull out bits of it to assess the effect of things. First, the effect of weight (for example for females of age 46.8, though it is the same idea for all the other age-sex combinations as well):

```
pp %>% filter(age == 46.8, sex == "female")
```

```
##   weight  age    sex  pred.old       pred
## 1    9.0 46.8 female 0.2994990 0.55303814
## 2   10.4 46.8 female 0.2370788 0.42818659
## 3   11.8 46.8 female 0.1842462 0.27804272
## 4   13.2 46.8 female 0.1410112 0.14407377
## 5   14.6 46.8 female 0.1065959 0.05884193
## 6   16.0 46.8 female 0.0797997 0.01935264
```

The predicted probabilities start off a lot higher and finish a fair bit lower. What about the effect of age? Let's take males of weight 9:

```
pp %>% filter(weight == 9, sex == "male")
```

```
##   weight   age  sex   pred.old      pred
## 1      9  26.0 male 0.06560117 0.1066453
## 2      9  46.8 male 0.08369818 0.2674861
## 3      9  67.6 male 0.10622001 0.4412181
## 4      9  88.4 male 0.13391640 0.5468890
## 5      9 109.2 male 0.16748125 0.5660054
## 6      9 130.0 male 0.20744380 0.4990420
```

This one is rather interesting: first of all, the predictions are very different. Also, the old predictions went steadily up, but the new ones go up for a bit, level off and start coming back down again. This is because of the squared term: without it, the predicted probabilities have to keep going in the same direction, either up or down.

Which age, weight and sex of animals is under the most risk under our new model? There's a trick here: work out the maximum value of pred, and then pick out the row(s) where pred is equal to that maximum value:

```
pp %>% filter(pred == max(pred))
```

```
##   weight   age    sex  pred.old      pred
## 1      9 109.2 female 0.4849694 0.8154681
```

Old, non-heavy females, but curiously enough not the *oldest* females (because of that squared term).

This would also work:

```
pp %>% summarize(max.pred = max(pred))
```

```
##    max.pred
## 1 0.8154681
```

but then you wouldn't see what contributed to that highest predicted probability. What you might do then is to say "let's find all the predicted probabilities bigger than 0.8":

```
pp %>% filter(pred > 0.8)
```

```
##   weight   age    sex  pred.old      pred
## 1      9  88.4 female 0.4198664 0.8035256
## 2      9 109.2 female 0.4849694 0.8154681
```

which includes the highest and second-highest ones. Or, you can *sort* the predictions in descending order and look at the first few:

```
pp %>% arrange(desc(pred)) %>% slice(1:10)
```

```
##    weight   age    sex  pred.old      pred
## 1     9.0 109.2 female 0.4849694 0.8154681
## 2     9.0  88.4 female 0.4198664 0.8035256
## 3     9.0 130.0 female 0.5505865 0.7714535
## 4     9.0  67.6 female 0.3574375 0.7279306
## 5    10.4 109.2 female 0.4063155 0.7278482
## 6    10.4  88.4 female 0.3447044 0.7122362
## 7    10.4 130.0 female 0.4710234 0.6713570
## 8    10.4  67.6 female 0.2879050 0.6182061
## 9    11.8 109.2 female 0.3321903 0.5790306
## 10    9.0 109.2   male 0.1674812 0.5660054
```

16.5 The brain of a cat

A large number (315) of psychology students were asked to imagine that they were serving on a university ethics committee hearing a complaint against animal research being done by a member of the faculty. The students were told that the surgery consisted of implanting a device called a cannula in each cat's brain, through which chemicals were introduced into the brain and the cats were then given psychological tests. At the end of the study, the cats' brains were subjected to histological analysis. The complaint asked that the researcher's authorization to carry out the study should be withdrawn, and the cats should be handed over to the animal rights group that filed the complaint. It was suggested that the research could just as well be done with computer simulations.

All of the psychology students in the survey were told all of this. In addition, they read a statement by the researcher that no animal felt much pain at any time, and that computer simulation was *not* an adequate substitute for animal research. Each student was also given *one* of the following scenarios that explained the benefit of the research:

- "cosmetic": testing the toxicity of chemicals to be used in new lines of hair care products.
- "theory": evaluating two competing theories about the function of a particular nucleus in the brain.
- "meat": testing a synthetic growth hormone said to potentially increase meat production.
- "veterinary": attempting to find a cure for a brain disease that is killing domesticated cats and endangered species of wild cats.

- "medical": evaluating a potential cure for a debilitating disease that afflicts many young adult humans.

Finally, each student completed two questionnaires: one that would assess their "relativism": whether or not they believe in universal moral principles (low score) or whether they believed that the appropriate action depends on the person and situation (high score). The second questionnaire assessed "idealism": a high score reflects a belief that ethical behaviour will always lead to good consequences (and thus that if a behaviour leads to any bad consequences at all, it is unethical).

After being exposed to all of that, each student stated their decision about whether the research should continue or stop.

I should perhaps stress at this point that no actual cats were harmed in the collection of these data (which can be found as a `.csv` file at link). The variables in the data set are these:

I get confused about the difference between morals and ethics. This is a very short description of that difference: urlhttp://smallbusiness.chron.com/differences-between-ethical-issues-moral-issues-business-48134.html. The basic idea is that morals are part of who you are, derived from religion, philosophy etc. Ethics are how you act in a particular situation: that is, your morals, what you believe, inform your ethics, what you do. That's why the students had to play the role of an "ethics committee", rather than a "morals committee"; presumably the researcher had good morals, but an ethics committee had to evaluate what he was planning to do.

- `decision`: whether the research should continue or stop (response)
- `idealism`: score on idealism questionnaire
- `relativism`: score on relativism questionnaire
- `gender` of student
- `scenario` of research benefits that the student read.

A more detailed discussion of this study is at link.

If you can believe it.

(a) Read in the data and check by looking at the structure of your data frame that you have something sensible. *Do not* call your data frame `decision`, since that's the name of one of the variables in it.

Solution

So, like this, using the name `decide` in my case:

```
my_url = "http://www.utsc.utoronto.ca/~butler/d29/decision.csv"
decide = read_csv(my_url)
```

```
## Parsed with column specification:
## cols(
##   decision = col_character(),
##   idealism = col_double(),
##   relativism = col_double(),
##   gender = col_character(),
##   scenario = col_character()
## )
```

```
decide
```

```
## # A tibble: 315 x 5
##    decision idealism relativism gender
##    <chr>       <dbl>      <dbl> <chr>
##  1 stop          8.2        5.1 Female
##  2 continue      6.8        5.3 Male
##  3 continue      8.2        6   Female
##  4 stop          7.4        6.2 Female
##  5 continue      1.7        3.1 Female
##  6 continue      5.6        7.7 Male
##  7 stop          7.2        6.7 Female
##  8 stop          7.8        4   Male
##  9 stop          7.8        4.7 Female
## 10 stop          8          7.6 Female
## # ... with 305 more rows, and 1 more
## #   variable: scenario <chr>
```

The variables are all the right things and of the right types: the decision, gender and the scenario are all text (representing categorical variables), and idealism and relativism, which were scores on a test, are quantitative (numerical). There are, as promised, 315 observations.

(b) Fit a logistic regression predicting `decision` from `gender`. Is there an effect of gender?

Solution

Turn the response into a `factor` somehow, either by creating a new variable in the data frame or like this:

```
decide.1 = glm(factor(decision) ~ gender, data = decide,
    family = "binomial")
summary(decide.1)
```

```
##
## Call:
## glm(formula = factor(decision) ~ gender, family = "binomial",
##     data = decide)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -1.5518  -1.0251   0.8446   0.8446   1.3377
##
## Coefficients:
##             Estimate Std. Error z value
## (Intercept)   0.8473     0.1543   5.491
## genderMale   -1.2167     0.2445  -4.976
```

```
##             Pr(>|z|)
## (Intercept) 3.99e-08 ***
## genderMale  6.50e-07 ***
## ---
## Signif. codes:
##   0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1  ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 425.57  on 314  degrees of freedom
## Residual deviance: 399.91  on 313  degrees of freedom
## AIC: 403.91
##
## Number of Fisher Scoring iterations: 4
```

The P-value for gender is 6.5×10^{-7}, which is very small, so there is definitely an effect of gender. It's not immediately clear what kind of effect it is: that's the reason for the next part, and we'll revisit this slope coefficient in a moment. Categorical *explanatory* variables are perfectly all right as text.

(c) To investigate the effect (or non-effect) of `gender`, create a contingency table by feeding `decision` and `gender` into `table`. What does this tell you?

Solution

```
with(decide, table(decision, gender))
```

```
##           gender
## decision   Female Male
##   continue     60   68
##   stop        140   47
```

Females are more likely to say that the study should stop (a clear majority), while males are more evenly split, with a small majority in favour of the study continuing.

If you want the column percents as well, you can use `prop.table`. Two steps: save the table from above into a variable, then feed *that* into `prop.table`, calling for column percentages rather than row percentages:

```
tab = with(decide, table(decision, gender))
prop.table(tab, 2)
```

```
##           gender
## decision      Female      Male
```

```
##   continue 0.3000000 0.5913043
##   stop     0.7000000 0.4086957
```

Why column percentages? Well, thinking back to STAB22 or some such place, when one of your variables is acting like a response or outcome (`decision` here), make the percentages out of the *other* one. Given that a student is a female, how likely are they to call for the research to stop? The other way around makes less sense: given that a person wanted the research to stop, how likely are they to be female?

About 70% of females and 40% of males want the research to stop. That's a giant-sized difference. No wonder it was significant.

The other way of making the table is to use `xtabs`, with the same result:

```
xtabs(~decision + gender, data = decide)
```

```
##           gender
## decision   Female Male
##   continue     60   68
##   stop        140   47
```

In this one, the frequency variable goes on the left side of the squiggle. We don't have one here (each row of the data frame represents one student), so we leave the left side blank. I tried putting a `.` there, but that doesn't work since there is no "whatever was there before" as there is, for example, in `update`.

(d)* Is your slope for `gender` in the previous logistic regression positive or negative? Is it applying to males or to females? Looking at the conclusions from your contingency table, what probability does that mean your logistic regression is actually modelling?

Solution

My slope is -1.2167, negative, and it is attached to males (note that the slope is called `gendermale`: because "female" is before "male" alphabetically, females are used as the baseline and this slope says how males compare to them). This negative male coefficient means that the probability of whatever is being modelled is *less* for males than it is for females. Looking at the contingency table for the last part, the probability of "stop" should be less for males, so the logistic regression is actually modelling the probability of "stop". Another way to reason that this must be the right answer is that the two values of `decision` are `continue` and `stop`; `continue` is first alphabetically, so it's the baseline, and the *other* one, `stop`, is the one whose probability is being modelled. That's why I made you do that contingency table. Another way to think about this is to do a prediction, which would go like this:

```
genders = c("Female", "Male")
new = tibble(gender = genders)
p = predict(decide.1, new, type = "response")
cbind(new, p)
```

```
##   gender         p
## 1 Female 0.7000000
## 2   Male 0.4086957
```

The probability of whatever-it-is is exactly 70% for females and about 40% for males. A quick look at the contingency table shows that exactly 70% (140/200) of the females think the research should stop, and a bit less than 50% of the males think the same thing. So the model is predicting the probability of "stop".

There's a logic to this: it's not just this way "because it is". It's the same idea of the first category, now of the response factor, being a "baseline", and what actually gets modelled is the *second* category, relative to the baseline.

(e) Add the two variables `idealism` and `relativism` to your logistic regression. Do either or both of them add significantly to your model? Explain briefly.

Solution

The obvious way of doing this is to type out the entire model, with the two new variables on the end. You have to remember to turn `decision` into a `factor` again:

```
decide.2 = glm(factor(decision) ~ gender + idealism +
    relativism, data = decide, family = "binomial")
summary(decide.2)
```

```
##
## Call:
## glm(formula = factor(decision) ~ gender + idealism + relativism,
##     family = "binomial", data = decide)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -2.2226  -0.9891   0.4798   0.8748   2.0442
##
## Coefficients:
##             Estimate Std. Error z value
## (Intercept)  -1.4876     0.9787  -1.520
## genderMale   -1.1710     0.2679  -4.372
## idealism      0.6893     0.1115   6.180
```

```
## relativism   -0.3432     0.1245  -2.757
##             Pr(>|z|)
## (Intercept)  0.12849
## genderMale  1.23e-05 ***
## idealism    6.41e-10 ***
## relativism   0.00584 **
## ---
## Signif. codes:
##   0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1  ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 425.57  on 314  degrees of freedom
## Residual deviance: 346.50  on 311  degrees of freedom
## AIC: 354.5
##
## Number of Fisher Scoring iterations: 4
```

This is not so bad, copying and pasting. But the way I like better, when you're making a smallish change to a longish model, is to use `update`:

```
decide.2 = update(decide.1, . ~ . + idealism +
    relativism)
summary(decide.2)
```

```
##
## Call:
## glm(formula = factor(decision) ~ gender + idealism + relativism,
##     family = "binomial", data = decide)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -2.2226  -0.9891   0.4798   0.8748   2.0442
##
## Coefficients:
##             Estimate Std. Error z value
## (Intercept)  -1.4876     0.9787  -1.520
## genderMale   -1.1710     0.2679  -4.372
## idealism      0.6893     0.1115   6.180
## relativism   -0.3432     0.1245  -2.757
##             Pr(>|z|)
## (Intercept)  0.12849
## genderMale  1.23e-05 ***
## idealism    6.41e-10 ***
```

```
## relativism   0.00584 **
## ---
## Signif. codes:
##   0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1  ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 425.57  on 314  degrees of freedom
## Residual deviance: 346.50  on 311  degrees of freedom
## AIC: 354.5
##
## Number of Fisher Scoring iterations: 4
```

Either way is good. The conclusion you need to draw is that they both have something to add, because their P-values are both less than 0.05.

(f) Add the variable `scenario` to your model. That is, fit a new model with that variable plus all the others.

Solution
To my mind, `update` wins hands down here:

```
decide.3 = update(decide.2, . ~ . + scenario)
```

You can display the summary here if you like, but we're not going to look at it yet.

(g) Use `anova` to compare the models with and without `scenario`. You'll have to add a `test="Chisq"` to your `anova`, to make sure that the test gets done. Does `scenario` make a difference or not, at $\alpha = 0.10$? Explain briefly. (The reason we have to do it this way is that `scenario` is a factor with five levels, so it has four slope coefficients. To test them all at once, which is what we need to make an overall test for `scenario`, this is the way it has to be done.)

Solution
These are the models that you fit in the last two parts:

```
anova(decide.2, decide.3, test = "Chisq")
```

```
## Analysis of Deviance Table
##
## Model 1: factor(decision) ~ gender + idealism + relativism
## Model 2: factor(decision) ~ gender + idealism + relativism + scenario
##   Resid. Df Resid. Dev Df Deviance Pr(>Chi)
```

```
## 1       311     346.50
## 2       307     338.06  4   8.4431  0.07663
##
## 1
## 2 .
## ---
## Signif. codes:
##   0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1  ' ' 1
```

The P-value is not less than 0.05, but it *is* less than 0.10, which is what I implied to assess it with, so the scenario does make some kind of difference.

Extra: another way to do this, which I like better (but the `anova` way was what I asked in the original question), is to look at `decide.3` and ask "what can I get rid of", in such a way that categorical variables stay or go as a whole. This is done using `drop1`. It's a little different from the corresponding thing in regression because the right way to do the test is not an F test, but now a chi-squared test (this is true for all generalized linear models of which logistic regression is one):

```r
drop1(decide.3, test = "Chisq")
```

```
## Single term deletions
##
## Model:
## factor(decision) ~ gender + idealism + relativism + scenario
##            Df Deviance    AIC    LRT
## <none>          338.06 354.06
## gender      1   359.61 373.61 21.546
## idealism    1   384.40 398.40 46.340
## relativism  1   344.97 358.97  6.911
## scenario    4   346.50 354.50  8.443
##             Pr(>Chi)
## <none>
## gender     3.454e-06 ***
## idealism   9.943e-12 ***
## relativism  0.008567 **
## scenario    0.076630 .
## ---
## Signif. codes:
##   0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1  ' ' 1
```

The test for `scenario` has four degrees of freedom (since there are five scenarios), and is in fact exactly the same test as in `anova`, significant at $\alpha = 0.10$.

(h) Look at the `summary` of your model that contained `scenario`. Bearing in mind that the slope coefficient for `scenariocosmetic` is zero (since this is the first scenario alphabetically), which scenarios have the most positive and most negative slope coefficients? What does that tell you about those scenarios' effects?

Solution

All right. This is the model I called `decide.3`:

```
summary(decide.3)
```

```
##
## Call:
## glm(formula = factor(decision) ~ gender + idealism + relativism +
##     scenario, family = "binomial", data = decide)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -2.3350  -0.9402   0.4645   0.8266   2.1564
##
## Coefficients:
##                    Estimate Std. Error
## (Intercept)         -1.5694     1.0426
## genderMale          -1.2551     0.2766
## idealism             0.7012     0.1139
## relativism          -0.3264     0.1267
## scenariomeat         0.1565     0.4283
## scenariomedical     -0.7095     0.4202
## scenariotheory       0.4501     0.4271
## scenarioveterinary  -0.1672     0.4159
##                    z value Pr(>|z|)
## (Intercept)         -1.505   0.1322
## genderMale          -4.537 5.70e-06 ***
## idealism             6.156 7.48e-10 ***
## relativism          -2.576   0.0100 *
## scenariomeat         0.365   0.7149
## scenariomedical     -1.688   0.0914 .
## scenariotheory       1.054   0.2919
## scenarioveterinary  -0.402   0.6878
## ---
## Signif. codes:
##   0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1  ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
```

```
##     Null deviance: 425.57  on 314  degrees of freedom
## Residual deviance: 338.06  on 307  degrees of freedom
## AIC: 354.06
##
## Number of Fisher Scoring iterations: 4
```

The most positive coefficient is for `theory` and the most negative one is for `medical`. (The zero coefficient is in the middle.) Since we are modelling the probability of saying that the research should *stop* (part (here)), this means that:

- the "theory" scenario (evaluating theories about brain function) is most likely to lead to someone saying that the research should stop (other things being equal)
- the "medical" scenario (finding a cure for a human disease) is most likely to lead to someone saying that the research should continue (or least likely to say that it should stop), again, other things being equal.

These make some kind of sense because being exposed to a scenario where there are tangible benefits later ought to be most favourable to the research continuing, and people are not going to be impressed by something that is "only theoretical" without any clear benefits. This also lends itself to a `predict` solution, but it's a little fiddly. I need some "average" values for the other variables, and I don't know what they are yet:

```
decide %>% summarize_if(is.numeric, median)
```

```
## # A tibble: 1 x 2
##   idealism relativism
##      <dbl>      <dbl>
## 1      6.5        6.1
```

("if the variable is numeric, get me its median".)

The other thing we need is a list of all the scenarios. Here's a cute way to do that:

```
scenarios = decide %>% count(scenario) %>% pull(scenario)
scenarios
```

```
## [1] "cosmetic"   "meat"       "medical"
## [4] "theory"     "veterinary"
```

I didn't really need to know how many observations there were for each scenario, but it was a handy way to find out which scenarios

there were without listing them all (and then having to pick out the unique ones).

So now let's make a data frame to predict from that has all scenarios for, let's say, females. (We don't have an interaction, so according to the model, the pattern is the same for males and females. So I could just as well have looked at males.)

```
new = crossing(idealism = 6.5, relativism = 6.1,
    gender = "Female", scenario = scenarios)
new
```

```
## # A tibble: 5 x 4
##   idealism relativism gender scenario
##      <dbl>      <dbl> <chr>  <chr>
## 1      6.5        6.1 Female cosmetic
## 2      6.5        6.1 Female meat
## 3      6.5        6.1 Female medical
## 4      6.5        6.1 Female theory
## 5      6.5        6.1 Female veterinary
```

```
p = predict(decide.3, new, type = "response")
cbind(new, p)
```

```
##   idealism relativism gender   scenario
## 1      6.5        6.1 Female   cosmetic
## 2      6.5        6.1 Female       meat
## 3      6.5        6.1 Female    medical
## 4      6.5        6.1 Female     theory
## 5      6.5        6.1 Female veterinary
##           p
## 1 0.7304565
## 2 0.7601305
## 3 0.5713725
## 4 0.8095480
## 5 0.6963099
```

This echoes what we found before: the probability of saying that the research should stop is highest for "theory" and the lowest for "medical".

I assumed in my model that the effect of the scenarios was the same for males and females. If I wanted to test that, I'd have to add an interaction and test that. This works most nicely using `update` and then `anova`, to fit the model with interaction and compare it with the model without:

```
decide.4 = update(decide.3, . ~ . + gender * scenario)
anova(decide.3, decide.4, test = "Chisq")
```

```
## Analysis of Deviance Table
##
## Model 1: factor(decision) ~ gender + idealism + relativism + scenario
## Model 2: factor(decision) ~ gender + idealism + relativism + scenario +
##     gender:scenario
##   Resid. Df Resid. Dev Df Deviance Pr(>Chi)
## 1       307     338.06
## 2       303     337.05  4   1.0122   0.9079
```

No evidence at all that the scenarios have different effects for the different genders. The appropriate `predict` should show that too:

```
genders = decide %>% count(gender) %>% pull(gender)
new = crossing(idealism = 6.5, relativism = 6.1,
    gender = genders, scenario = scenarios)
new
```

```
## # A tibble: 10 x 4
##    idealism relativism gender scenario
##       <dbl>      <dbl> <chr>  <chr>
##  1      6.5        6.1 Female cosmetic
##  2      6.5        6.1 Female meat
##  3      6.5        6.1 Female medical
##  4      6.5        6.1 Female theory
##  5      6.5        6.1 Female veterinary
##  6      6.5        6.1 Male   cosmetic
##  7      6.5        6.1 Male   meat
##  8      6.5        6.1 Male   medical
##  9      6.5        6.1 Male   theory
## 10      6.5        6.1 Male   veterinary
```

```
p = predict(decide.4, new, type = "response")
cbind(new, p)
```

```
##    idealism relativism gender   scenario
## 1       6.5        6.1 Female   cosmetic
## 2       6.5        6.1 Female       meat
## 3       6.5        6.1 Female    medical
## 4       6.5        6.1 Female     theory
## 5       6.5        6.1 Female veterinary
## 6       6.5        6.1   Male   cosmetic
## 7       6.5        6.1   Male       meat
## 8       6.5        6.1   Male    medical
## 9       6.5        6.1   Male     theory
## 10      6.5        6.1   Male veterinary
##            p
```

```
## 1  0.7670133
## 2  0.7290272
## 3  0.5666034
## 4  0.8251472
## 5  0.6908189
## 6  0.3928582
## 7  0.5438679
## 8  0.2860504
## 9  0.5237310
## 10 0.4087458
```

The probability of "stop" is a lot higher for females than for males (that is the strong `gender` effect we found earlier), but the *pattern* is about the same for males and females: the difference in probabilities is about the same, and also both genders have almost the highest predicted probability for `theory` (the highest male one is actually for `meat` but there's not much in it) and the lowest one for `medical`, as we found before. The pattern is not *exactly* the same because these predictions came from the model with interaction. If you use the model without interaction, you get this:

Strictly, we should look at the difference in log-odds.

```
p = predict(decide.3, new, type = "response")
cbind(new, p)
```

```
##    idealism relativism gender   scenario
## 1       6.5        6.1 Female   cosmetic
## 2       6.5        6.1 Female       meat
## 3       6.5        6.1 Female    medical
## 4       6.5        6.1 Female     theory
## 5       6.5        6.1 Female veterinary
## 6       6.5        6.1   Male   cosmetic
## 7       6.5        6.1   Male       meat
## 8       6.5        6.1   Male    medical
## 9       6.5        6.1   Male     theory
## 10      6.5        6.1   Male veterinary
##            p
## 1  0.7304565
## 2  0.7601305
## 3  0.5713725
## 4  0.8095480
## 5  0.6963099
## 6  0.4358199
## 7  0.4745996
## 8  0.2753530
## 9  0.5478510
## 10 0.3952499
```

and the scenarios are ranked in *exactly* the same order by males and females. The probabilities from the two models are very close (compare them), so there is no value in adding the interaction, as we found before.

So fitting an interaction was a waste of time, but it was worth checking whether it was.

(i) Describe the effects that having (i) a higher idealism score and (ii) a higher relativity score have on a person's probability of saying that the research should stop. Do each of these increase or decrease that probability? Explain briefly.

Solution

Look back at the summary for the model that I called `decide.3`. (Or `decide.2`: the slope coefficients are very similar.) The one for `idealism` is positive, so that a higher idealism score goes with a greater likelihood of saying that the research should stop. The slope coefficient for `relativity` is negative, so it's the other way around: a higher relativity score goes with a *lower* chance of saying that the research should stop. That's all I needed, but as an extra, we can look back at the description of these scales in the question. The `relativism` one was that a person believed that the most moral action depends on the situation (as opposed to a person having something like religious faith that asserts universal moral principles that are always true. That would be a low score on the relativism scale). Somebody with a low score on this scale might believe something like "it is always wrong to experiment on animals", whereas somebody with a high relativism score might say that it was sometimes justified. Thus, other things being equal, a low relativism score would go with "stop" and a high relativism score would (or might) go with "continue". This would mean a *negative* slope coefficient for `relativism`, which is what we observed. (Are you still with me? There was some careful thinking there.) What about `idealism`? This is a belief that ethical behaviour will always lead to good consequences, and thus, if the consequences are bad, the behaviour must not have been ethical. A person who scores high on idealism is likely to look at the consequences (experimentation on animals), see that as a bad thing, and thus conclude that the research should be stopped. The `idealism` slope coefficient, by that argument, should be positive, and is. This will also lead to a `predict`. We need "low" and "high" scores on the idealism and relativism tests. Let's use the quartiles:

```
decide %>% summarize(i_q1 = quantile(idealism,
    0.25), i_q3 = quantile(idealism, 0.75), r_q1 = quantile(relativism,
    0.25), r_q3 = quantile(relativism, 0.75))
```

```
## # A tibble: 1 x 4
##    i_q1  i_q3  r_q1  r_q3
##   <dbl> <dbl> <dbl> <dbl>
## 1   5.6   7.5   5.4   6.8
```

There is a more elegant way, but this works. Let's use the scenario `cosmetic` that was middling in its effects, and think about females:

```
new = crossing(idealism = c(5.6, 7.5), relativism = c(5.4,
    6.8), gender = "Female", scenario = "cosmetic")
new
```

```
## # A tibble: 4 x 4
##   idealism relativism gender scenario
##      <dbl>      <dbl> <chr>  <chr>
## 1      5.6        5.4 Female cosmetic
## 2      5.6        6.8 Female cosmetic
## 3      7.5        5.4 Female cosmetic
## 4      7.5        6.8 Female cosmetic
```

```
p = predict(decide.3, new, type = "response")
cbind(new, p)
```

```
##   idealism relativism gender scenario
## 1      5.6        5.4 Female cosmetic
## 2      5.6        6.8 Female cosmetic
## 3      7.5        5.4 Female cosmetic
## 4      7.5        6.8 Female cosmetic
##           p
## 1 0.6443730
## 2 0.5342955
## 3 0.8728724
## 4 0.8129966
```

For both of the idealism scores, the higher relativism score went with a lower probability of "stop" (the "negative" effect), and for both of the relativism scores, the higher idealism score went with a *higher* probability of "stop" (the positive effect).

That's quite enough discussion of the question, except that the data didn't come to me in the form that you see them, so I figured I would like to share the story of the data processing as well. I think this is important because in your future work you are likely to spend a lot of your time getting data from how you receive it to something suitable for analysis.

These data came from a psychology study (with, probably, the students in a class serving as experimental subjects). Social scientists

like to use SPSS software, so the data came to me as an SPSS `.sav` file. The least-fuss way of handling this that I could think of was to use `import` from the `rio` package, which I think I mentioned before:

If you took STAB23, you'll have used PSPP, which is a free version of SPSS.

```
library(rio)
x = import("/home/ken/Downloads/Logistic.sav")
str(x)
```

```
## 'data.frame':    315 obs. of  11 variables:
##  $ decision   : num  0 1 1 0 1 1 0 0 0 0 ...
##   ..- attr(*, "format.spss")= chr "F1.0"
##   ..- attr(*, "labels")= Named num  0 1
##   .. ..- attr(*, "names")= chr  "stop" "continue"
##  $ idealism   : num  8.2 6.8 8.2 7.4 1.7 5.6 7.2 7.8 7.8 8 ...
##   ..- attr(*, "format.spss")= chr "F12.4"
##  $ relatvsm   : num  5.1 5.3 6 6.2 3.1 7.7 6.7 4 4.7 7.6 ...
##   ..- attr(*, "format.spss")= chr "F12.4"
##  $ gender     : num  0 1 0 0 0 1 0 1 0 0 ...
##   ..- attr(*, "format.spss")= chr "F1.0"
##   ..- attr(*, "labels")= Named num  0 1
##   .. ..- attr(*, "names")= chr  "Female" "Male"
##  $ cosmetic   : num  1 1 1 1 1 1 1 1 1 1 ...
##   ..- attr(*, "format.spss")= chr "F1.0"
##  $ theory     : num  0 0 0 0 0 0 0 0 0 0 ...
##   ..- attr(*, "format.spss")= chr "F1.0"
##  $ meat       : num  0 0 0 0 0 0 0 0 0 0 ...
##   ..- attr(*, "format.spss")= chr "F1.0"
##  $ veterin    : num  0 0 0 0 0 0 0 0 0 0 ...
##   ..- attr(*, "format.spss")= chr "F1.0"
##  $ idealism_LN: num  2.104 1.917 2.104 2.001 0.531 ...
##   ..- attr(*, "format.spss")= chr "F8.2"
##   ..- attr(*, "display_width")= int 13
##  $ relatvsm_LN: num  1.63 1.67 1.79 1.82 1.13 ...
##   ..- attr(*, "format.spss")= chr "F8.2"
##   ..- attr(*, "display_width")= int 13
##  $ scenario   : num  1 1 1 1 1 1 1 1 1 1 ...
##   ..- attr(*, "format.spss")= chr "F8.2"
##   ..- attr(*, "display_width")= int 10
```

The last line `str` displays the "structure" of the data frame that was obtained. Normally a data frame read into R has a much simpler structure than this, but this is R trying to interpret how SPSS does things. Here, each column (listed on the lines beginning with a dollar sign) has some values, listed on the `atomic` line and possibly some "attributes", which affect how the variable is interpreted. The

Which always makes me think of https://www.youtube.com/watch?v=1Tko1G6XRiQ.

important one for us is the "names" attribute; if a column has this, it needs to be interpreted as categorical with the names as levels (going in order with the values in the `atomic` line). Thus, on the `gender` line, the subjects are a female (0), then a male (1), then three females, then a male, and so on. Variables like `gender` are thus so far neither really factors nor text variables, and so we'll have to do a bit of processing before we can use them: we want to replace the numerical values by the appropriate "level".

To turn a numeric variable into text depending on the value, we can use `ifelse`, but this gets unwieldy if there are more than two values to translate. For that kind of job, I think `case_when` is a lot easier to read. It also lets us have a catch-all for catching errors — "impossible" values occur distressingly often in real data:

```
xx = x %>% mutate(decision = case_when(decision ==
    0 ~ "stop", decision == 1 ~ "continue", TRUE ~
    "error"), gender = case_when(gender == 0 ~
    "Female", gender == 1 ~ "Male", TRUE ~ "error"),
    scenario = case_when(scenario == 1 ~ "cosmetic",
        scenario == 2 ~ "theory", scenario ==
            3 ~ "meat", scenario == 4 ~ "veterinary",
        scenario == 5 ~ "medical", TRUE ~ "error"))
xx %>% as_tibble() %>% select(-(cosmetic:veterin))
```

```
## # A tibble: 315 x 7
##    decision idealism relatvsm gender
##    <chr>       <dbl>    <dbl> <chr>
##  1 stop          8.2      5.1 Female
##  2 continue      6.8      5.3 Male
##  3 continue      8.2      6   Female
##  4 stop          7.4      6.2 Female
##  5 continue      1.7      3.1 Female
##  6 continue      5.6      7.7 Male
##  7 stop          7.2      6.7 Female
##  8 stop          7.8      4   Male
##  9 stop          7.8      4.7 Female
## 10 stop          8        7.6 Female
## # ... with 305 more rows, and 3 more
## #   variables: idealism_LN <dbl>,
## #   relatvsm_LN <dbl>, scenario <chr>
```

`xx` is a "real" `data.frame` (that's what `rio` reads in), and has some extra columns that we don't want to see right now.

I have three new variables being created in one `mutate`. Each is being created using a `case_when`. The thing on the left of each squiggle

is a logical condition being tested; the first of these logical conditions to come out `TRUE` provides the value for the new variable on the right of the squiggle. Thus, if the (old) `scenario` is 2, the new `scenario` will be `theory`. The `TRUE` lines in each case provide something that is guaranteed to be true, even if all the other lines are false (eg. if `scenario` is actually recorded as 7, which would be an error).

I overwrote the old variable values with the new ones, which is a bit risky, but then I'd have more things to get rid of later.

My next step is to check that I don't actually have any errors:

```
xx %>% count(scenario, gender, decision)
```

```
## # A tibble: 20 x 4
##    scenario   gender decision     n
##    <chr>      <chr>  <chr>    <int>
##  1 cosmetic   Female continue     8
##  2 cosmetic   Female stop        26
##  3 cosmetic   Male   continue    17
##  4 cosmetic   Male   stop        11
##  5 meat       Female continue    11
##  6 meat       Female stop        31
##  7 meat       Male   continue    12
##  8 meat       Male   stop         9
##  9 medical    Female continue    19
## 10 medical    Female stop        24
## 11 medical    Male   continue    15
## 12 medical    Male   stop         5
## 13 theory     Female continue     8
## 14 theory     Female stop        30
## 15 theory     Male   continue    12
## 16 theory     Male   stop        14
## 17 veterinary Female continue    14
## 18 veterinary Female stop        29
## 19 veterinary Male   continue    12
## 20 veterinary Male   stop         8
```

Don't see any errors there.

So now let's write what we have to a file. I think a `.csv` would be smart:

```
xx %>% select(decision, idealism, relatvsm, gender,
    scenario) %>% write_csv("decision.csv")
```

There is one more tiny detail: in SPSS, variable names can have a maximum of eight letters. "Relativism" has 10. So the original data file had the name "relativism" minus the two "i"s. I changed that so

you would be dealing with a proper English word. (That change is not shown here.)

There is actually a *town* called Catbrain. It's in England, near Bristol, and seems to be home to a street of car dealerships. Just to show that you can do *anything* in R:

```
library(ggmap)
gg = get_map("Catbrain", zoom = 14, maptype = "roadmap")
ggmap(gg)
```

(fix this up later)

The car dealerships are along Lysander Road. Change the `zoom` to a bigger number to see them. Evidently, there is no other Catbrain anywhere in the world, since this is the only one that Google Maps knows about.

The name, according to link, means "rough stony soil", from Middle English, and has nothing to do with cats or their brains at all.

This picture is actually a `ggplot`, so you can do things like plotting points on the map as well:

```
lons = c(-2.6, -2.62)
lats = c(51.51, 51.52, 51.53)
points = expand.grid(lon = lons, lat = lats)
ggmap(gg) + geom_point(data = points, aes(x = lon,
    y = lat))
```

Note the unusual way in which you specify the points to plot. `ggmap` replaces the usual `ggplot` which is where the default data frame and `aes` for the plot would normally live, so you have to specify them in `geom_point` as shown.

I have no idea why you'd want to plot those points. Normally, on a map, you'd find the longitude and latitude of some named points (a process known as "geocoding").

The M5 is one of the English "motorways" (like 400-series highways or US Interstates). The M5 goes from Birmingham to Exeter. You can tell that this is England because of the huge number of traffic circles, known there as "roundabouts". One of the first things they teach you in British driving schools is how to handle roundabouts: which lane to approach them in, which (stick-shift) gear to be in, and when you're supposed to signal where you're going. I hope I still remember all that for when I next drive in England!

16.6 How not to get heart disease

What is associated with heart disease? In a study, a large number of variables were measured, as follows:

- `age` (years)
- `sex` male or female
- `pain.type` Chest pain type (4 values: typical angina, atypical angina, non-anginal pain, asymptomatic)
- `resting.bp` Resting blood pressure, on admission to hospital
- `serum.chol` Serum cholesterol
- `high.blood.sugar`: greater than 120, yes or no
- `electro` resting electrocardiographic results (normal, having ST-T, hypertrophy)
- `max.hr` Maximum heart rate
- `angina` Exercise induced angina (yes or no)
- `oldpeak` ST depression induced by exercise relative to rest. See link.
- `slope` Slope of peak exercise ST segment. Sloping up, flat or sloping down
- `colored` number of major vessels (0–3) coloured by fluoroscopy
- `thal` normal, fixed defect, reversible defect
- `heart.disease` 1=absent, 2=present

I don't know what most of those are, but we will not let that stand in our way. Our aim is to find out what variables are associated with heart disease, and what values of those variables give high probabilities of heart disease being present. The data are in link.

(a) Read in the data. Display the first few lines and convince yourself that those values are reasonable.

Solution
A `.csv` file, so:

```
my_url = "http://www.utsc.utoronto.ca/~butler/d29/heartf.csv"
heart = read_csv(my_url)
```

```
## Warning: Missing column names filled in:
## 'X1' [1]
```

```
## Parsed with column specification:
## cols(
##   X1 = col_integer(),
##   age = col_integer(),
##   sex = col_character(),
##   pain.type = col_character(),
##   resting.bp = col_integer(),
##   serum.chol = col_integer(),
##   high.blood.sugar = col_character(),
##   electro = col_character(),
##   max.hr = col_integer(),
##   angina = col_character(),
##   oldpeak = col_double(),
##   slope = col_character(),
##   colored = col_integer(),
##   thal = col_character(),
##   heart.disease = col_character()
## )
```

```
heart
```

```
## # A tibble: 270 x 15
##       X1   age sex   pain.type resting.bp
##    <int> <int> <chr> <chr>          <int>
##  1     1    70 male  asymptom~        130
##  2     2    67 fema~ nonangin~        115
##  3     3    57 male  atypical         124
##  4     4    64 male  asymptom~        128
##  5     5    74 fema~ atypical         120
##  6     6    65 male  asymptom~        120
##  7     7    56 male  nonangin~        130
##  8     8    59 male  asymptom~        110
##  9     9    60 male  asymptom~        140
## 10    10    63 fema~ asymptom~        150
## # ... with 260 more rows, and 10 more
## #   variables: serum.chol <int>,
## #   high.blood.sugar <chr>, electro <chr>,
## #   max.hr <int>, angina <chr>,
## #   oldpeak <dbl>, slope <chr>,
## #   colored <int>, thal <chr>,
## #   heart.disease <chr>
```

This display is kind of compressed. Every time you see a squiggle, as in `pain~`, that means there is actually more (not shown), so what you do is to check that what you see here matches at least the start of what is supposed to be there.

You should check that the variables that should be numbers actually are, that the variables that should be categorical have (as far as is shown) the right values as per my description above, and you should make some comment in that direction.

My variables appear to be correct, apart possibly for that variable `X1` which is actually just the row number.

(b) In a logistic regression, what probability will be predicted here? Explain briefly but convincingly. (Is each line of the data file one observation or a summary of several?)

Solution

Each line of the data file is a single observation, not frequencies of yes and no (like the premature babies question, later, is). The response variable is a factor, so the first level is the baseline and the *second* level is the one predicted. R puts factor levels alphabetically, so `no` is first and `yes` is second. That is, a logistic regression will predict the probability that a person *does* have heart disease. I want to see that logic (which is why I said "convincingly"): one observation per line, and therefore that the second level of the factor is predicted, which is `yes`.

(c)* Fit a logistic regression predicting heart disease from everything else (if you have a column called `X` or `X1`, ignore that), and display the results.

Solution

A lot of typing, since there are so many variables. Don't forget that the response variable *must* be a factor:

```
heart.1 = glm(factor(heart.disease) ~ age + sex +
    pain.type + resting.bp + serum.chol + high.blood.sugar +
    electro + max.hr + angina + oldpeak + slope +
    colored + thal, family = "binomial", data = heart)
```

You can split this over several lines (and probably should), but make sure to end each line in such a way that there is unambiguously more to come, for example with a plus or a comma (though probably the fact that you have an unclosed bracket will be enough).

The output is rather lengthy:

```
summary(heart.1)
```

```
##
## Call:
## glm(formula = factor(heart.disease) ~ age + sex + pain.type +
##     resting.bp + serum.chol + high.blood.sugar + electro + max.hr +
##     angina + oldpeak + slope + colored + thal, family = "binomial",
```

```
##     data = heart)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -2.6431  -0.4754  -0.1465   0.3342   2.8100
##
## Coefficients:
##                      Estimate Std. Error
## (Intercept)         -3.973837   3.133311
## age                 -0.016007   0.026394
## sexmale              1.763012   0.580761
## pain.typeatypical   -0.997298   0.626233
## pain.typenonanginal -1.833394   0.520808
## pain.typetypical    -2.386128   0.756538
## resting.bp           0.026004   0.012080
## serum.chol           0.006621   0.004228
## high.blood.sugaryes -0.370040   0.626396
## electronormal       -0.633593   0.412073
## electroSTT           0.013986   3.184512
## max.hr              -0.019337   0.011486
## anginayes            0.596869   0.460540
## oldpeak              0.449245   0.244631
## slopeflat            0.827054   0.966139
## slopeupsloping      -0.122787   1.041666
## colored              1.199839   0.280947
## thalnormal           0.146197   0.845517
## thalreversible       1.577988   0.838550
##                     z value Pr(>|z|)
## (Intercept)          -1.268 0.204707
## age                  -0.606 0.544208
## sexmale               3.036 0.002400 **
## pain.typeatypical    -1.593 0.111264
## pain.typenonanginal  -3.520 0.000431 ***
## pain.typetypical     -3.154 0.001610 **
## resting.bp            2.153 0.031346 *
## serum.chol            1.566 0.117322
## high.blood.sugaryes  -0.591 0.554692
## electronormal        -1.538 0.124153
## electroSTT            0.004 0.996496
## max.hr               -1.683 0.092278 .
## anginayes             1.296 0.194968
## oldpeak               1.836 0.066295 .
## slopeflat             0.856 0.391975
## slopeupsloping       -0.118 0.906166
```

```
## colored               4.271 1.95e-05 ***
## thalnormal            0.173 0.862723
## thalreversible        1.882 0.059863 .
## ---
## Signif. codes:
##   0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1  ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 370.96  on 269  degrees of freedom
## Residual deviance: 168.90  on 251  degrees of freedom
## AIC: 206.9
##
## Number of Fisher Scoring iterations: 6
```

I didn't ask you for further comment, but note that quite a lot of these variables are factors, so you get slopes for things like `pain.typeatypical`. When you have a factor in a model, there is a slope for each level except for the first, which is a baseline (and its slope is taken to be zero). That would be `asymptomatic` for `pain.type`. The *t*-tests for the other levels of `pain.type` say whether that level of pain type differs significantly (in terms of probability of heart disease) from the baseline level. Here, pain type `atypical` is not significantly different from the baseline, but the other two pain types, `nonanginal` and `typical`, *are* significantly different. If you think about this from an ANOVA-like point of view, the question about `pain.type`'s significance is really "is there at least one of the pain types that is different from the others", and if we're thinking about whether we should keep `pain.type` in the logistic regression, this is the kind of question we should be thinking about.

(d) Quite a lot of our explanatory variables are factors. To assess whether the factor as a whole should stay or can be removed, looking at the slopes won't help us very much (since they tell us whether the other levels of the factor differ from the baseline, which may not be a sensible comparison to make). To assess which variables are candidates to be removed, factors included (properly), we can use `drop1`. Feed `drop1` a fitted model and the words `test="Chisq"` (take care of the capitalization!) and you'll get a list of P-values. Which variable is the one that you would remove first? Explain briefly.

Solution
Following the instructions:

```r
drop1(heart.1, test = "Chisq")
```

```
## Single term deletions
##
## Model:
## factor(heart.disease) ~ age + sex + pain.type + resting.bp +
##     serum.chol + high.blood.sugar + electro + max.hr + angina +
##     oldpeak + slope + colored + thal
##                  Df Deviance    AIC     LRT
## <none>                168.90 206.90
## age               1   169.27 205.27  0.3705
## sex               1   179.16 215.16 10.2684
## pain.type         3   187.85 219.85 18.9557
## resting.bp        1   173.78 209.78  4.8793
## serum.chol        1   171.34 207.34  2.4484
## high.blood.sugar  1   169.25 205.25  0.3528
## electro           2   171.31 205.31  2.4119
## max.hr            1   171.84 207.84  2.9391
## angina            1   170.55 206.55  1.6562
## oldpeak           1   172.44 208.44  3.5449
## slope             2   172.98 206.98  4.0844
## colored           1   191.78 227.78 22.8878
## thal              2   180.78 214.78 11.8809
##                   Pr(>Chi)
## <none>
## age              0.5427474
## sex              0.0013533 **
## pain.type        0.0002792 ***
## resting.bp       0.0271810 *
## serum.chol       0.1176468
## high.blood.sugar 0.5525052
## electro          0.2994126
## max.hr           0.0864608 .
## angina           0.1981121
## oldpeak          0.0597303 .
## slope            0.1297422
## colored          1.717e-06 ***
## thal             0.0026308 **
## ---
## Signif. codes:
##   0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1  ' ' 1
```

The highest P-value, 0.5525, goes with `high.blood.sugar`, so this one comes out first. (The P-value for `age` is almost as high, 0.5427, so you might guess that this will be next.)

You might be curious about how these compare with the P-values

on `summary`. These two P-values are almost the same as the ones on `summary`, because they are a two-level factor and a numeric variable respectively, and so the tests are equivalent in the two cases. (The P-values are not identical because the tests on `summary` and `drop1` are the kind of thing that would be identical on a regular regression but are only "asymptotically the same" in logistic regression, so you'd expect them to be close without being the same, as here. "Asymptotically the same" means that if you had an infinitely large sample size, they'd be identical, but our sample size of 200-odd individuals is not infinitely large! Anyway, the largest P-value on the `summary` is 0.9965, which goes with `electroSTT`. `electro`, though, is a factor with three levels; this P-value says that `STT` is almost identical (in its effects on heart disease) with the baseline `hypertrophy`. But there is a third level, `normal`, which is a bit different from `hypertrophy`. So the factor `electro` overall has some effect on heart disease, which is reflected in the `drop1` P-value of 0.12: this might go later, but it has to stay for now because at least one of its levels is different from the others in its effect on heart disease. (In backward elimination, multi-level factors are removed in their entirety if *none* of their levels have a different effect from any of the others.)

The power just went out here, so I am using my laptop on battery on its own screen, rather than on the big screen I have in my office, which is much better.

(e) I'm not going to make you do the whole backward elimination (I'm going to have you use `step` for that later), but do one step: that is, fit a model removing the variable you think should be removed, using `update`, and then run `drop1` again to see which variable will be removed next.

Solution

`update` is the obvious choice here, since we're making a small change to a *very* big model:

```
heart.2 = update(heart.1, . ~ . - high.blood.sugar)
drop1(heart.2, test = "Chisq")
```

```
## Single term deletions
##
## Model:
## factor(heart.disease) ~ age + sex + pain.type + resting.bp +
##     serum.chol + electro + max.hr + angina + oldpeak + slope +
##     colored + thal
##            Df Deviance    AIC     LRT
## <none>          169.25 205.25
## age         1   169.66 203.66  0.4104
```

```
## sex         1   179.27 213.27 10.0229
## pain.type   3   190.54 220.54 21.2943
## resting.bp  1   173.84 207.84  4.5942
## serum.chol  1   171.69 205.69  2.4458
## electro     2   171.65 203.65  2.3963
## max.hr      1   172.35 206.35  3.0969
## angina      1   170.78 204.78  1.5323
## oldpeak     1   173.26 207.26  4.0094
## slope       2   173.18 205.18  3.9288
## colored     1   191.87 225.87 22.6232
## thal        2   181.61 213.61 12.3588
##             Pr(>Chi)
## <none>
## age         0.521756
## sex         0.001546 **
## pain.type  9.145e-05 ***
## resting.bp  0.032080 *
## serum.chol  0.117841
## electro     0.301750
## max.hr      0.078440 .
## angina      0.215764
## oldpeak     0.045248 *
## slope       0.140240
## colored    1.971e-06 ***
## thal        0.002072 **
## ---
## Signif. codes:
##   0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1  ' ' 1
```

The power is back.

The next variable to go is indeed `age`, with a P-value that has hardly changed: it is now 0.5218.

(f) Use `step` to do a backward elimination to find which variables have an effect on heart disease. Display your final model (which you can do by saving the output from `step` in a variable, and asking for the summary of that. In `step`, you'll need to specify a starting model (the one from part (here)), the direction of elimination, and the test to base the elimination decision on (the same one as you used in `drop1`).

Solution

The hints ought to lead you to this:

```
heart.3 = step(heart.1, direction = "backward",
    test = "Chisq")
```

```
## Start:  AIC=206.9
## factor(heart.disease) ~ age + sex + pain.type + resting.bp +
##     serum.chol + high.blood.sugar + electro + max.hr + angina +
##     oldpeak + slope + colored + thal
##
##                     Df Deviance    AIC
## - high.blood.sugar  1   169.25 205.25
## - age               1   169.27 205.27
## - electro           2   171.31 205.31
## - angina            1   170.55 206.55
## <none>                  168.90 206.90
## - slope             2   172.98 206.98
## - serum.chol        1   171.34 207.34
## - max.hr            1   171.84 207.84
## - oldpeak           1   172.44 208.44
## - resting.bp        1   173.78 209.78
## - thal              2   180.78 214.78
## - sex               1   179.16 215.16
## - pain.type         3   187.85 219.85
## - colored           1   191.78 227.78
##                        LRT  Pr(>Chi)
## - high.blood.sugar  0.3528 0.5525052
## - age               0.3705 0.5427474
## - electro           2.4119 0.2994126
## - angina            1.6562 0.1981121
## <none>
## - slope             4.0844 0.1297422
## - serum.chol        2.4484 0.1176468
## - max.hr            2.9391 0.0864608 .
## - oldpeak           3.5449 0.0597303 .
## - resting.bp        4.8793 0.0271810 *
## - thal             11.8809 0.0026308 **
## - sex              10.2684 0.0013533 **
## - pain.type        18.9557 0.0002792 ***
## - colored          22.8878 1.717e-06 ***
## ---
## Signif. codes:
##   0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1  ' ' 1
##
## Step:  AIC=205.25
## factor(heart.disease) ~ age + sex + pain.type + resting.bp +
##     serum.chol + electro + max.hr + angina + oldpeak + slope +
##     colored + thal
##
```

```
##              Df Deviance    AIC     LRT
## - electro     2   171.65 203.65  2.3963
## - age         1   169.66 203.66  0.4104
## - angina      1   170.78 204.78  1.5323
## - slope       2   173.18 205.18  3.9288
## <none>            169.25 205.25
## - serum.chol  1   171.69 205.69  2.4458
## - max.hr      1   172.35 206.35  3.0969
## - oldpeak     1   173.26 207.26  4.0094
## - resting.bp  1   173.84 207.84  4.5942
## - sex         1   179.27 213.27 10.0229
## - thal        2   181.61 213.61 12.3588
## - pain.type   3   190.54 220.54 21.2943
## - colored     1   191.87 225.87 22.6232
##               Pr(>Chi)
## - electro     0.301750
## - age         0.521756
## - angina      0.215764
## - slope       0.140240
## <none>
## - serum.chol  0.117841
## - max.hr      0.078440 .
## - oldpeak     0.045248 *
## - resting.bp  0.032080 *
## - sex         0.001546 **
## - thal        0.002072 **
## - pain.type  9.145e-05 ***
## - colored    1.971e-06 ***
## ---
## Signif. codes:
##   0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1  ' ' 1
##
## Step:  AIC=203.65
## factor(heart.disease) ~ age + sex + pain.type + resting.bp +
##     serum.chol + max.hr + angina + oldpeak + slope + colored +
##     thal
##
##              Df Deviance    AIC     LRT
## - age         1   172.03 202.03  0.3894
## - angina      1   173.13 203.13  1.4843
## <none>            171.65 203.65
## - slope       2   175.99 203.99  4.3442
## - max.hr      1   175.00 205.00  3.3560
## - serum.chol  1   175.11 205.11  3.4610
```

```
## - oldpeak     1   175.42 205.42  3.7710
## - resting.bp  1   176.61 206.61  4.9639
## - thal        2   182.91 210.91 11.2633
## - sex         1   182.77 212.77 11.1221
## - pain.type   3   192.83 218.83 21.1859
## - colored     1   194.90 224.90 23.2530
##               Pr(>Chi)
## - age        0.5326108
## - angina     0.2231042
## <none>
## - slope      0.1139366
## - max.hr     0.0669599 .
## - serum.chol 0.0628319 .
## - oldpeak    0.0521485 .
## - resting.bp 0.0258824 *
## - thal       0.0035826 **
## - sex        0.0008531 ***
## - pain.type  9.632e-05 ***
## - colored    1.420e-06 ***
## ---
## Signif. codes:
##   0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1  ' ' 1
##
## Step:  AIC=202.03
## factor(heart.disease) ~ sex + pain.type + resting.bp + serum.chol +
##     max.hr + angina + oldpeak + slope + colored + thal
##
##              Df Deviance    AIC     LRT
## - angina      1   173.57 201.57  1.5385
## <none>            172.03 202.03
## - slope       2   176.33 202.33  4.2934
## - max.hr      1   175.00 203.00  2.9696
## - serum.chol  1   175.22 203.22  3.1865
## - oldpeak     1   175.92 203.92  3.8856
## - resting.bp  1   176.63 204.63  4.5911
## - thal        2   183.38 209.38 11.3500
## - sex         1   183.97 211.97 11.9388
## - pain.type   3   193.71 217.71 21.6786
## - colored     1   195.73 223.73 23.6997
##               Pr(>Chi)
## - angina     0.2148451
## <none>
## - slope      0.1168678
## - max.hr     0.0848415 .
```

```
## - serum.chol 0.0742492 .
## - oldpeak    0.0487018 *
## - resting.bp 0.0321391 *
## - thal       0.0034306 **
## - sex        0.0005498 ***
## - pain.type  7.609e-05 ***
## - colored    1.126e-06 ***
## ---
## Signif. codes:
##   0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1  ' ' 1
##
## Step:  AIC=201.57
## factor(heart.disease) ~ sex + pain.type + resting.bp + serum.chol +
##     max.hr + oldpeak + slope + colored + thal
##
##              Df Deviance    AIC     LRT
## <none>            173.57 201.57
## - slope       2   178.44 202.44  4.8672
## - serum.chol  1   176.83 202.83  3.2557
## - max.hr      1   177.52 203.52  3.9442
## - oldpeak     1   177.79 203.79  4.2135
## - resting.bp  1   178.56 204.56  4.9828
## - thal        2   186.22 210.22 12.6423
## - sex         1   185.88 211.88 12.3088
## - pain.type   3   200.68 222.68 27.1025
## - colored     1   196.98 222.98 23.4109
##               Pr(>Chi)
## <none>
## - slope      0.0877201 .
## - serum.chol 0.0711768 .
## - max.hr     0.0470322 *
## - oldpeak    0.0401045 *
## - resting.bp 0.0256006 *
## - thal       0.0017978 **
## - sex        0.0004508 ***
## - pain.type  5.603e-06 ***
## - colored    1.308e-06 ***
## ---
## Signif. codes:
##   0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1  ' ' 1
```

The output is very long, and I made it tiny so that you would see all of it. In terms of AIC, which is what `step` uses, `age` hangs on for a bit, but eventually gets eliminated.

There are a lot of variables left.

(g) `Display the summary of the model that came out of 'step'.`

Solution
This:

```
summary(heart.3)
```

```
##
## Call:
## glm(formula = factor(heart.disease) ~ sex + pain.type + resting.bp +
##     serum.chol + max.hr + oldpeak + slope + colored + thal, family = "binomial",
##     data = heart)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -2.7103  -0.4546  -0.1442   0.3864   2.7121
##
## Coefficients:
##                      Estimate Std. Error
## (Intercept)         -4.818418   2.550437
## sexmale              1.850559   0.561583
## pain.typeatypical   -1.268233   0.604488
## pain.typenonanginal -2.086204   0.486591
## pain.typetypical    -2.532340   0.748941
## resting.bp           0.024125   0.011077
## serum.chol           0.007142   0.003941
## max.hr              -0.020373   0.010585
## oldpeak              0.467028   0.233280
## slopeflat            0.859564   0.922749
## slopeupsloping      -0.165832   0.991474
## colored              1.134561   0.261547
## thalnormal           0.323543   0.813442
## thalreversible       1.700314   0.805127
##                     z value Pr(>|z|)
## (Intercept)          -1.889 0.058858 .
## sexmale               3.295 0.000983 ***
## pain.typeatypical    -2.098 0.035903 *
## pain.typenonanginal  -4.287 1.81e-05 ***
## pain.typetypical     -3.381 0.000722 ***
## resting.bp            2.178 0.029410 *
## serum.chol            1.812 0.069966 .
## max.hr               -1.925 0.054262 .
## oldpeak               2.002 0.045284 *
## slopeflat             0.932 0.351582
```

```
## slopeupsloping       -0.167 0.867167
## colored               4.338 1.44e-05 ***
## thalnormal            0.398 0.690818
## thalreversible        2.112 0.034699 *
## ---
## Signif. codes:
##   0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1  ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 370.96  on 269  degrees of freedom
## Residual deviance: 173.57  on 256  degrees of freedom
## AIC: 201.57
##
## Number of Fisher Scoring iterations: 6
```

Not all of the P-values in the `step` output wound up being less than 0.05, but they are all at least reasonably small. As discussed above, some of the P-values in the `summary` are definitely *not* small, but they go with factors where there are significant effects *somewhere*. For example, `thalnormal` is not significant (that is, `normal` is not significantly different from the baseline `fixed`), but the other level `reversible` *is* different from `fixed`. You might be wondering about `slope`: on the `summary` there is nothing close to significance, but on the `step` output, `slope` has at least a reasonably small P-value of 0.088. This is because the significant difference does not involve the baseline: it's actually between `flat` with a positive slope and `upsloping` with a negative one.

(h) We are going to make a large number of predictions. Create a data frame that contains all combinations of representative values for all the variables in the model that came out of `step`. By "representative" I mean all the values for a categorical variable, and the first and third quartiles for a numeric variable.

Solution

Let's take a breath first. There are two pieces of stuff to do here: we need to get the quartiles of the quantitative variables, and we need all the different values of the categorical ones. There are several of each, so we want some kind of automated method. The easy way of getting the quartiles is via `summary` of the whole data frame:

```
summary(heart)
```

```
##        X1              age
##  Min.   :  1.00   Min.   :29.00
```

```
##  1st Qu.: 68.25   1st Qu.:48.00
##  Median :135.50   Median :55.00
##  Mean   :135.50   Mean   :54.43
##  3rd Qu.:202.75   3rd Qu.:61.00
##  Max.   :270.00   Max.   :77.00
##      sex             pain.type
##  Length:270         Length:270
##  Class :character   Class :character
##  Mode  :character   Mode  :character
##
##
##
##    resting.bp      serum.chol
##  Min.   : 94.0   Min.   :126.0
##  1st Qu.:120.0   1st Qu.:213.0
##  Median :130.0   Median :245.0
##  Mean   :131.3   Mean   :249.7
##  3rd Qu.:140.0   3rd Qu.:280.0
##  Max.   :200.0   Max.   :564.0
##  high.blood.sugar     electro
##  Length:270         Length:270
##  Class :character   Class :character
##  Mode  :character   Mode  :character
##
##
##
##      max.hr         angina
##  Min.   : 71.0   Length:270
##  1st Qu.:133.0   Class :character
##  Median :153.5   Mode  :character
##  Mean   :149.7
##  3rd Qu.:166.0
##  Max.   :202.0
##     oldpeak        slope
##  Min.   :0.00   Length:270
##  1st Qu.:0.00   Class :character
##  Median :0.80   Mode  :character
##  Mean   :1.05
##  3rd Qu.:1.60
##  Max.   :6.20
##     colored           thal
##  Min.   :0.0000   Length:270
##  1st Qu.:0.0000   Class :character
##  Median :0.0000   Mode  :character
```

```
##  Mean   :0.6704
##  3rd Qu.:1.0000
##  Max.   :3.0000
##  heart.disease
##  Length:270
##  Class :character
##  Mode  :character
##
##
##
```

This is the old-fashioned "base R" way of doing it. The `tidyverse` way is to get the Q1 and Q3 of just the variables that are numeric. To that, we'll write little functions to get Q1 and Q3 of anything, and then use `summarize_if` to apply those to the quantitative variables:

```
q1 = function(x) quantile(x, 0.25)
q3 = function(x) quantile(x, 0.75)
heart %>% summarize_if(is.numeric, funs(q1, q3))
```

```
## # A tibble: 1 x 14
##   X1_q1 age_q1 resting.bp_q1 serum.chol_q1
##   <dbl>  <dbl>         <dbl>         <dbl>
## 1  68.2     48           120           213
## # ... with 10 more variables:
## #   max.hr_q1 <dbl>, oldpeak_q1 <dbl>,
## #   colored_q1 <dbl>, X1_q3 <dbl>,
## #   age_q3 <dbl>, resting.bp_q3 <dbl>,
## #   serum.chol_q3 <dbl>, max.hr_q3 <dbl>,
## #   oldpeak_q3 <dbl>, colored_q3 <dbl>
```

These are actually all the Q1s followed by all the Q3s, but it's hard to see that because the column names got truncated (every time you see a squiggle it means that there is more not shown). One way to see the results is to "transpose" the result with the variable-quartile combinations in a column and the actual quartile values in another:

```
heart2 = heart %>% summarize_if(is.numeric, funs(q1,
    q3)) %>% gather(vq, quartile, everything())
```

```
## Warning: attributes are not identical across measure variables;
## they will be dropped
```

```
heart2
```

```
## # A tibble: 14 x 2
##    vq            quartile
```

```
##    <chr>            <dbl>
##  1 X1_q1             68.2
##  2 age_q1            48
##  3 resting.bp_q1    120
##  4 serum.chol_q1    213
##  5 max.hr_q1        133
##  6 oldpeak_q1         0
##  7 colored_q1         0
##  8 X1_q3            203.
##  9 age_q3            61
## 10 resting.bp_q3    140
## 11 serum.chol_q3    280
## 12 max.hr_q3        166
## 13 oldpeak_q3         1.6
## 14 colored_q3         1
```

If you want to be really fancy:

```
heart %>% summarize_if(is.numeric, funs(q1, q3)) %>%
    gather(vq, quartile, everything()) %>% separate(vq,
    c("variable", "q"), "_") %>% spread(q, quartile)
```

```
## Warning: attributes are not identical across measure variables;
## they will be dropped
```

```
## # A tibble: 7 x 3
##   variable      q1    q3
##   <chr>      <dbl> <dbl>
## 1 age         48    61
## 2 colored      0     1
## 3 max.hr     133   166
## 4 oldpeak      0     1.6
## 5 resting.bp 120   140
## 6 serum.chol 213   280
## 7 X1          68.2 203.
```

The logic is (beyond what we had above) splitting up `vq` into the two things it really is (I had to put in the `"_"` at the end because it would otherwise also split up at the dots), and then I deliberately untidy the column that contains `q1` and `q3` into two columns of those names.

The categorical variables are a bit trickier, because they will have different numbers of possible values. Here's my idea:

```
heart %>% select_if(is.character) %>% mutate_all(factor) %>%
    summary()
```

```
##      sex             pain.type
##  female: 87   asymptomatic:129
##  male  :183   atypical    : 42
##               nonanginal  : 79
##               typical     : 20
##  high.blood.sugar        electro
##  no :230          hypertrophy:137
##  yes: 40          normal     :131
##                   STT        :  2
##
##  angina            slope
##  no :181   downsloping: 18
##  yes: 89   flat       :122
##            upsloping  :130
##
##          thal     heart.disease
##  fixed     : 14   no :150
##  normal    :152   yes:120
##  reversible:104
##
```

My thought was that if you pass a genuine factor into `summary` (as opposed to a categorical variable that is text), it displays all its "levels" (different categories). So, to get to that point, I had to select all the categorical-as-text variables (which is actually all the ones that are not numeric), and then make a factor out of each of them. `mutate_all` does the same thing to all the columns: it runs `factor` on them, and saves the results back in variables of the same name as they were before. Using `summary` also shows how many observations we had in each category.

What I would really like to do is to save these category names somewhere so I don't have to type them below when I make all possible combinations. My go at that:

```
heart3 = heart %>% select_if(is.character) %>%
    gather(vname, value, everything()) %>% distinct()
heart3 %>% print(n = Inf)
```

```
## # A tibble: 21 x 2
##    vname            value
##    <chr>            <chr>
##  1 sex              male
##  2 sex              female
##  3 pain.type        asymptomatic
##  4 pain.type        nonanginal
```

```
##  5 pain.type        atypical
##  6 pain.type        typical
##  7 high.blood.sugar no
##  8 high.blood.sugar yes
##  9 electro          hypertrophy
## 10 electro          normal
## 11 electro          STT
## 12 angina           no
## 13 angina           yes
## 14 slope            flat
## 15 slope            upsloping
## 16 slope            downsloping
## 17 thal             normal
## 18 thal             reversible
## 19 thal             fixed
## 20 heart.disease    yes
## 21 heart.disease    no
```

Gosh, that was easier than I thought. A *lot* easier. The technique is a lot like that idea for making facetted plots of something against "all the *x*s": you gather everything up into one column containing a variable name and another column containing the variable value. This contains a lot of repeats; the `distinct` keeps one of each and throws away the rest.

Hmm, another way would be to count everything:

```
heart3 = heart %>% select_if(is.character) %>%
    gather(vname, value, everything()) %>% count(vname,
    value)
heart3 %>% print(n = Inf)
```

```
## # A tibble: 21 x 3
##    vname            value            n
##    <chr>            <chr>        <int>
##  1 angina           no             181
##  2 angina           yes             89
##  3 electro          hypertrophy    137
##  4 electro          normal         131
##  5 electro          STT              2
##  6 heart.disease    no             150
##  7 heart.disease    yes            120
##  8 high.blood.sugar no             230
##  9 high.blood.sugar yes             40
## 10 pain.type        asymptomatic   129
## 11 pain.type        atypical        42
```

```
## 12 pain.type        nonanginal      79
## 13 pain.type        typical         20
## 14 sex              female          87
## 15 sex              male           183
## 16 slope            downsloping     18
## 17 slope            flat           122
## 18 slope            upsloping      130
## 19 thal             fixed           14
## 20 thal             normal         152
## 21 thal             reversible     104
```

That perhaps is more obvious in retrospect, because there are no new tools there.

Now, we come to setting up the variables that we are going to make all combinations of. For a quantitative variable such as `age`, I need to go back to `heart2`:

```
heart2
```

```
## # A tibble: 14 x 2
##    vq            quartile
##    <chr>            <dbl>
##  1 X1_q1             68.2
##  2 age_q1            48
##  3 resting.bp_q1    120
##  4 serum.chol_q1    213
##  5 max.hr_q1        133
##  6 oldpeak_q1         0
##  7 colored_q1         0
##  8 X1_q3            203.
##  9 age_q3            61
## 10 resting.bp_q3    140
## 11 serum.chol_q3    280
## 12 max.hr_q3        166
## 13 oldpeak_q3         1.6
## 14 colored_q3         1
```

and pull out the rows whose names contain `age_`. This is done using `str_detect` from `stringr`. Here's how it goes for `age`:

If you're selecting *columns*, you can use select-helpers, but for rows, not.

Which is loaded with the *tidyverse* so you don't have to load it.

```
heart2 %>% filter(str_detect(vq, "age_"))
```

```
## # A tibble: 2 x 2
##   vq     quartile
##   <chr>     <dbl>
## 1 age_q1       48
## 2 age_q3       61
```

and one more step to get just the quartiles:

```
heart2 %>% filter(str_detect(vq, "age_")) %>%
    pull(quartile)
```

```
## [1] 48 61
```

We'll be doing this a few times, so we should write a function to do it:

```
get_quartiles = function(d, x) {
    d %>% filter(str_detect(vq, x)) %>% pull(quartile)
}
```

and to test:

```
get_quartiles(heart2, "age_")
```

```
## [1] 48 61
```

Yep. I put the underscore in so as to not catch other variables that have `age` inside them but not at the end.

For the categorical variables, we need to look in `heart3`:

```
heart3
```

```
## # A tibble: 21 x 3
##    vname            value            n
##    <chr>            <chr>        <int>
##  1 angina           no             181
##  2 angina           yes             89
##  3 electro          hypertrophy    137
##  4 electro          normal         131
##  5 electro          STT              2
##  6 heart.disease    no             150
##  7 heart.disease    yes            120
##  8 high.blood.sugar no             230
##  9 high.blood.sugar yes             40
## 10 pain.type        asymptomatic   129
## # ... with 11 more rows
```

then choose the rows with the right thing in `vname`, and then pull just the `value` column. This is sufficiently like the previous one that I think we can write a function right away:

```
get_categories = function(d, x) {
    d %>% filter(vname == x) %>% pull(value)
}
get_categories(heart3, "electro")
```

```
## [1] "hypertrophy" "normal"      "STT"
```

All right, setup, using my usual habit of plural names, and using those functions we just wrote:

```
sexes = get_categories(heart3, "sex")
pain.types = get_categories(heart3, "pain.type")
resting.bps = get_quartiles(heart2, "resting.bp_")
serum.chols = get_quartiles(heart2, "serum.chol_")
max.hrs = get_quartiles(heart2, "max.hr_")
oldpeaks = get_quartiles(heart2, "oldpeak_")
slopes = get_categories(heart3, "slope")
coloreds = get_quartiles(heart2, "colored_")
thals = get_categories(heart3, "thal")
```

All combos of all of those (and there will be a lot of those):

```
heart.new = crossing(sex = sexes, pain.type = pain.types,
    resting.bp = resting.bps, serum.chol = serum.chols,
    max.hr = max.hrs, oldpeak = oldpeaks, slope = slopes,
    colored = coloreds, thal = thals)
heart.new
```

```
## # A tibble: 2,304 x 9
##    sex   pain.type resting.bp serum.chol
##    <chr> <chr>          <dbl>      <dbl>
##  1 fema~ asymptom~        120        213
##  2 fema~ asymptom~        120        213
##  3 fema~ asymptom~        120        213
##  4 fema~ asymptom~        120        213
##  5 fema~ asymptom~        120        213
##  6 fema~ asymptom~        120        213
##  7 fema~ asymptom~        120        213
##  8 fema~ asymptom~        120        213
##  9 fema~ asymptom~        120        213
## 10 fema~ asymptom~        120        213
## # ... with 2,294 more rows, and 5 more
## #   variables: max.hr <dbl>, oldpeak <dbl>,
## #   slope <chr>, colored <dbl>, thal <chr>
```

Yeah, that's a lot. Fortunately, we won't have to look at them all.

(i) Obtain the predicted probabilities of heart disease for the data frame you constructed in the last part, using your model that came out of `step`. Add these predictions to the data frame from the previous part (as a column in that data frame).

Solution

Get the predictions, which is less scary than it seems:

```
p = predict(heart.3, heart.new, type = "response")
```

and the easiest way to add these to `heart.new` is this:

```
heart.new$pred = p
```

Or, if you like, with a `mutate` of this kind:

```
heart.x = heart.new %>% mutate(prediction = p)
```

Let's take a look at a few of the predictions, by way of sanity-checking:

```
heart.new %>% sample_n(8)
```

```
## # A tibble: 8 x 10
##   sex   pain.type resting.bp serum.chol
##   <chr> <chr>          <dbl>      <dbl>
## 1 fema~ asymptom~        120        213
## 2 fema~ asymptom~        140        213
## 3 fema~ atypical         120        280
## 4 male  asymptom~        120        213
## 5 male  nonangin~        120        280
## 6 fema~ nonangin~        120        280
## 7 male  nonangin~        140        213
## 8 male  asymptom~        140        213
## # ... with 6 more variables: max.hr <dbl>,
## #   oldpeak <dbl>, slope <chr>,
## #   colored <dbl>, thal <chr>, pred <dbl>
```

This seems at least reasonably sane.

(j) Find the largest predicted probability (which is the predicted probability of heart disease) and display all the variables that it was a prediction for.

Solution

This can be done in one step:

```
heart.new %>% filter(pred == max(pred))
```

```
## # A tibble: 1 x 10
##   sex   pain.type resting.bp serum.chol
##   <chr> <chr>          <dbl>      <dbl>
## 1 male  asymptom~        140        280
## # ... with 6 more variables: max.hr <dbl>,
## #   oldpeak <dbl>, slope <chr>,
## #   colored <dbl>, thal <chr>, pred <dbl>
```

or if you didn't think of that, you can find the maximum first, and then display the rows with predictions close to it:

```
heart.new %>% summarize(m = max(pred))
```

```
## # A tibble: 1 x 1
##       m
##   <dbl>
## 1 0.984
```

```
heart.new %>% filter(pred > 0.98)
```

```
## # A tibble: 1 x 10
##   sex   pain.type resting.bp serum.chol
##   <chr> <chr>          <dbl>      <dbl>
## 1 male  asymptom~        140        280
## # ... with 6 more variables: max.hr <dbl>,
## #   oldpeak <dbl>, slope <chr>,
## #   colored <dbl>, thal <chr>, pred <dbl>
```

or even find *which* row has the maximum, and then display that row:

```
heart.new %>% summarize(row = which.max(pred))
```

```
## # A tibble: 1 x 1
##     row
##   <int>
## 1  1398
```

```
heart.new %>% slice(1398)
```

```
## # A tibble: 1 x 10
##   sex   pain.type resting.bp serum.chol
##   <chr> <chr>          <dbl>      <dbl>
## 1 male  asymptom~        140        280
## # ... with 6 more variables: max.hr <dbl>,
## #   oldpeak <dbl>, slope <chr>,
## #   colored <dbl>, thal <chr>, pred <dbl>
```

or sort the rows by `pred`, descending, and display the top few:

```
heart.new %>% arrange(desc(pred)) %>% print(n = 8)
```

```
## # A tibble: 2,304 x 10
##   sex   pain.type resting.bp serum.chol
##   <chr> <chr>          <dbl>      <dbl>
## 1 male  asymptom~        140        280
```

```
## 2 male  asymptom~        140        213
## 3 male  asymptom~        120        280
## 4 male  asymptom~        140        280
## 5 male  asymptom~        140        280
## 6 male  asymptom~        140        280
## 7 male  asymptom~        120        213
## 8 male  asymptom~        140        280
## # ... with 2,296 more rows, and 6 more
## #   variables: max.hr <dbl>, oldpeak <dbl>,
## #   slope <chr>, colored <dbl>, thal <chr>,
## #   pred <dbl>
```

(k) Compare the `summary` of the final model from `step` with your highest predicted heart disease probability and the values of the other variables that make it up. Are they consistent?

Solution

Since we were predicting the probability of heart disease, a more positive slope in the model from `step` will be associated with a higher probability of heart disease. So, there, we are looking for a couple of things: if the variable is a factor, we're looking for the level with the most positive slope (bearing in mind that this might be the baseline), and for a numeric variable, if the slope is positive, a *high* value is associated with heart disease, and if negative, a low value. Bearing that in mind, we go back to my `summary(heart.3)` and we have:

- `sex`: being male has the higher risk, by a lot
- `pain`: all the slopes shown are negative, so the highest risk goes with the baseline one `asymptomatic`.
- `resting.bp`: positive slope, so higher risk with higher value.
- `serum.chol`: same.
- `max.hr`: negative slope, so greatest risk with *smaller* value.
- `oldpeak`: positive slope, greater risk with higher value again.
- `slope`: `flat` has greatest risk.
- `colored`: positive slope, so beware of higher value.
- `thal`: `reversible` has greatest risk.

Then we can do the same thing for the prediction. For the numerical variables, we may need to check back to the previous part to see whether the value shown was high or low. Once you have done that,

you can see that the variable values for the highest predicted probability do indeed match the ones we thought should be the highest risk. The interesting thing about this is that after adjusting for all of the other variables, there is a greater risk of heart disease if you are male (and the model shows that the risk is *much* greater). That is to say, it's being male that makes the difference, not the fact that any of the other variables are different for males. Perhaps, therefore, the easiest way to avoid a heart attack is to not be male!

16.7 *Successful breastfeeding*

A regular pregnancy lasts 40 weeks, and a baby that is born at or before 33 weeks is called "premature". The number of weeks at which a baby is born is called its "gestational age". Premature babies are usually smaller than normal and may require special care. It is a good sign if, when the mother and baby leave the hospital to go home, the baby is successfully breastfeeding.

The data in link are from a study of 64 premature infants. There are three columns: the gestational age (a whole number of weeks), the number of babies of that gestational age that were successfully breastfeeding when they left the hospital, and the number that were not. (There were multiple babies of the same gestational age, so the 64 babies are summarized in 6 rows of data.)

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

Solution

No great challenge here, I hope:

```
my_url = "http://www.utsc.utoronto.ca/~butler/d29/breastfeed.csv"
breastfeed = read_csv(my_url)
```

```
## Parsed with column specification:
## cols(
##   gest.age = col_integer(),
##   bf.yes = col_integer(),
##   bf.no = col_integer()
## )
```

```
breastfeed
```

```
## # A tibble: 6 x 3
##   gest.age bf.yes bf.no
##      <int>  <int> <int>
## 1       28      2     4
## 2       29      2     3
```

```
## 3       30      7     2
## 4       31      7     2
## 5       32     16     4
## 6       33     14     1
```

That looks reasonable.

(b) Verify that there were indeed 64 infants, by having R do a suitable calculation on your data frame that gives the right answer for the right reason.

Solution

The second and third columns are all frequencies, so it's a question of adding them up. For example:

```
breastfeed %>% summarize(total = sum(bf.yes) +
    sum(bf.no))
```

```
## # A tibble: 1 x 1
##   total
##   <int>
## 1    64
```

or if you want to go nuts (this one gathers all the frequencies together into one column and then adds them up):

```
breastfeed %>% gather(yesno, freq, bf.yes:bf.no) %>%
    summarize(total = sum(freq))
```

```
## # A tibble: 1 x 1
##   total
##   <int>
## 1    64
```

Find a way to get it done. If it works and it does the right thing, it's good.

Do *not* copy the numbers out of your data frame, type them in again and use R to add them up. Do something with your data frame as you read it in.

(c) Do you think, looking at the data, that there is a relationship between gestational age and whether or not the baby was successfully breastfeeding when it left the hospital? Explain briefly.

Solution

The babies with the youngest gestational age (the most premature) were mostly *not* breastfeeding when they left the hospital. Most of the 30- and 31-week babies were breastfeeding, and almost all of the

32- and 33-week babies were breastfeeding. So I think there will be a relationship: as gestational age increases, the probability that the baby will be breastfeeding will also increase. (This, looking ahead, suggests a positive slope in a logistic regression.)

(d) Why is logistic regression a sensible technique to use here? Explain briefly.

Solution

The response variable is a yes/no: whether or not an infant is breastfeeding. We want to predict the probability of the response being in one or the other category. This is what logistic regression does. (The explanatory variable(s) are usually numerical, as here, but they could be factors as well, or a mixture. The key is the kind of response. The number of babies that are successfully breastfeeding at a certain gestational age is modelled as binomial with n being the total number of babies of that gestational age, and p being something that might depend, and here *does* depend, on gestational age.)

(e) Fit a logistic regression to predict the probability that an infant will be breastfeeding from its gestational age. Show the output from your logistic regression.

Solution

These are summarized data, rather than one infant per line, so what we have to do is to make a two-column response "matrix", successes in the first column and failures in the second, and then predict *that* from gestational age. (That's why this was three marks rather than two.) So, let's make the `response` first:

```
response = with(breastfeed, cbind(bf.yes, bf.no))
response
```

```
##      bf.yes bf.no
## [1,]      2     4
## [2,]      2     3
## [3,]      7     2
## [4,]      7     2
## [5,]     16     4
## [6,]     14     1
```

or, more Tidyverse-like, but we have to remember to turn it into a `matrix`:

```
response = breastfeed %>% select(starts_with("bf")) %>%
    as.matrix()
response
```

```
##      bf.yes bf.no
## [1,]      2     4
## [2,]      2     3
## [3,]      7     2
## [4,]      7     2
## [5,]     16     4
## [6,]     14     1
```

I used a select-helper, because what immediately came to me was that the names of the columns I wanted started with `bf`, but whatever way you have that works is good. Now we fit the logistic regression:

```
breastfeed.1 = glm(response ~ gest.age, data = breastfeed,
    family = "binomial")
summary(breastfeed.1)
```

```
##
## Call:
## glm(formula = response ~ gest.age, family = "binomial", data = breastfeed)
##
## Deviance Residuals:
##       1        2        3        4        5
## -0.1472  -0.4602   0.8779   0.1114  -0.6119
##       6
##  0.3251
##
## Coefficients:
##             Estimate Std. Error z value
## (Intercept) -16.7198     6.0630  -2.758
## gest.age      0.5769     0.1977   2.918
##             Pr(>|z|)
## (Intercept)  0.00582 **
## gest.age     0.00352 **
## ---
## Signif. codes:
##   0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1  ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 11.1772  on 5  degrees of freedom
## Residual deviance:  1.4968  on 4  degrees of freedom
## AIC: 19.556
##
## Number of Fisher Scoring iterations: 4
```

(f) Does the significance or non-significance of the slope of `gest.age`

surprise you? Explain briefly.

Solution

The slope *is* significant (P-value 0.0035 is much less than 0.05). We said above that we expected there to be a relationship between gestational age and whether or not the baby was breastfeeding, and this significant slope is confirming that there *is* a relationship. So this is exactly what we expected to see, and not a surprise at all. If you concluded above that you did *not* see a relationship, you should colour yourself surprised here. Consistency.

(g) Is your slope (in the `Estimate` column) positive or negative? What does that mean, in terms of gestational ages and breastfeeding? Explain briefly.

Solution

My slope is 0.5769, positive. That means that as the explanatory variable, gestational age, increases, the probability of the event (that the baby is breastfeeding) also increases. This is also what I observed above: almost all of the near-term (large gestational age) babies were breastfeeding, whereas a fair few of the small-gestational-age (very premature) ones were not.

(h) Obtain the predicted probabilities that an infant will successfully breastfeed for each of the gestational ages in the data set, and display them side by side with the observed data.

Solution

This is the easier version of `predict`, where you do not have to create a data frame of values to predict from (since you are using the original data). Thus, you only need something like this:

```
p = predict(breastfeed.1, type = "response")
cbind(breastfeed, p)
```

```
##   gest.age bf.yes bf.no         p
## 1       28      2     4 0.3620282
## 2       29      2     3 0.5025820
## 3       30      7     2 0.6427290
## 4       31      7     2 0.7620821
## 5       32     16     4 0.8508177
## 6       33     14     1 0.9103511
```

You can see that the predicted probabilities go steadily up as the gestational age goes up, just as we would have expected.

If you only wanted certain gestational ages, for example 25, 30 and 35, you would do that like this:

```
ages.new = data.frame(gest.age = c(25, 30, 35))
p2 = predict(breastfeed.1, ages.new, type = "response")
cbind(ages.new, p2)
```

```
##   gest.age         p2
## 1       25 0.09134905
## 2       30 0.64272897
## 3       35 0.96987261
```

Or, if you wanted to make a graph of the observed and predicted proportions/probabilities, you could do something like this:

```
breastfeed %>% mutate(total = bf.yes + bf.no,
    obs = bf.yes/total) %>% mutate(pred = p) %>%
    ggplot(aes(x = gest.age, y = obs)) + geom_line(aes(y = pred)) +
    geom_point(aes(size = total))
```

What did I do there? I first created some new variables: `count` is the total number of babies of each gestational age, `obs` is the observed proportion of breastfeeding babies at each gestational age (number of yes divided by total), and `pred` which are the predictions we did above. Then I make a plot on which I plot the observed proportions against gestational age (as points), and then I want to plot the predictions joined by lines. To do *that*, I need to change which y-variable I am plotting (now `pred` rather than `obs`), and the way I do that is to put an `aes` inside the `geom_line` to say "use the x I had before, but use this y instead". I also wanted to draw attention to the gestational ages where more babies were observed; I did this by making the *size* of the plotted points proportional to how many babies there were at that gestational age (which was the quantity `total` I calculated above). This took a couple of attempts to get right: I put `size=` in the

original `aes`, but I didn't realize that it controlled the line thickness as well (which looked really weird). So I moved the `geom_point` to the end and put the `size=` in there, to make sure that the size thing *only* applied to the points. The legend for `total` tells you what size point corresponds to how many total babies.

The idea is that the observed and predicted should be reasonably close, or at least not grossly different, and I think they *are* close, which indicates that our model is doing a good job.

16.8 Making it over the mountains

In 1846, the Donner party (Donner and Reed families) left Springfield, Illinois for California in covered wagons. After reaching Fort Bridger, Wyoming, the leaders decided to find a new route to Sacramento. They became stranded in the eastern Sierra Nevada mountains at a place now called Donner Pass, when the region was hit by heavy snows in late October. By the time the survivors were rescued on April 21, 1847, 40 out of 87 had died.

After the rescue, the age and gender of each person in the party was recorded, along with whether they survived or not. The data are in link.

(a) Read in the data and display its structure. Does that agree with the description in the previous paragraph?

Solution

Nothing very new here:

```
my_url = "http://www.utsc.utoronto.ca/~butler/d29/donner.txt"
donner = read_delim(my_url, " ")
```

```
## Parsed with column specification:
## cols(
##   age = col_integer(),
##   gender = col_character(),
##   survived = col_character()
## )
```

```
donner
```

```
## # A tibble: 45 x 3
##      age gender survived
##    <int> <chr>  <chr>
##  1    23 male   no
##  2    40 female yes
##  3    40 male   yes
```

```
##  4    30 male   no
##  5    28 male   no
##  6    40 male   no
##  7    45 female no
##  8    62 male   no
##  9    65 male   no
## 10    45 female no
## # ... with 35 more rows
```

The ages look like ages, and the other variables are categorical as promised, with the right levels. So this looks sensible.

I usually write (or rewrite) the data description myself, but the one I found for these data at link was so nice that I used it as is. You can see that the data format used on that website is not as nice as ours. (I did some work to make it look nicer for you, with proper named categories rather than 0 and 1.)

(b) Make graphical or numerical summaries for each pair of variables. That is, you should make a graph or numerical summary for each of `age` vs. `gender`, `age` vs. `survived` and `gender` vs. `survived`. In choosing the kind of graph or summary that you will use, bear in mind that `survived` and `gender` are factors with two levels.

Solution

Thinking about graphs first: we have two numeric-vs-factor graphs (the ones involving `age`), which I think should be boxplots, though they could be something like side-by-side histograms (if you are willing to grapple with `facet_grid`). The other two variables are both factors, so a good graph for them would be something like a grouped bar plot (as in the question about parasites earlier). If you prefer numerical summaries: you could do mean age (or some other summary of age) for each group defined by gender or survivorship, and you could do a cross-tabulation of frequencies for gender and survival. I'll take any mixture of graphs and numerical summaries that addresses each pair of variables somehow and summarizes them in a sensible way. Starting with `age` vs. `gender`:

```r
ggplot(donner, aes(x = gender, y = age)) + geom_boxplot()
```

or:

```
ggplot(donner, aes(x = age)) + geom_histogram(bins = 10) +
    facet_grid(gender ~ .)
```

or:

```
aggregate(age ~ gender, donner, mean)
```

```
##   gender      age
## 1 female 31.06667
## 2   male 32.16667
```

or:

```
donner %>% group_by(gender) %>% summarize(m = mean(age))
```

```
## # A tibble: 2 x 2
##   gender     m
```

```
##   <chr>  <dbl>
## 1 female  31.1
## 2 male    32.2
```

Age vs. `survived` is the same idea:

```
ggplot(donner, aes(x = survived, y = age)) + geom_boxplot()
```

or:

```
ggplot(donner, aes(x = age)) + geom_histogram(bins = 10) +
    facet_grid(survived ~ .)
```

or:

```
aggregate(age ~ survived, donner, mean)
```

```
##   survived   age
## 1       no 35.48
## 2      yes 27.20
```

or:

```
donner %>% group_by(survived) %>% summarize(m = mean(age))
```

```
## # A tibble: 2 x 2
##   survived     m
##   <chr>    <dbl>
## 1 no        35.5
## 2 yes       27.2
```

For `survived` against `gender`, the obvious thing is a cross-tabulation, gotten like this:

```
with(donner, table(gender, survived))
```

```
##         survived
## gender   no yes
##   female  5  10
##   male   20  10
```

or like this:

```
donner %>% group_by(gender, survived) %>% summarize(n = n())
```

```
## # A tibble: 4 x 3
## # Groups:   gender [?]
##   gender survived     n
##   <chr>  <chr>    <int>
## 1 female no           5
## 2 female yes         10
## 3 male   no          20
## 4 male   yes         10
```

For a graph, borrow the grouped bar-plot idea from the parasites question:

```
ggplot(donner, aes(x = gender, fill = survived)) +
    geom_bar(position = "dodge")
```

I think this way around for `x` and `fill` is better, since we want to ask something like "out of the females, how many survived" (that is, gender is explanatory and survival response).

(c) For each of the three graphs or summaries in the previous question, what do they tell you about the relationship between the pair of variables concerned? Explain briefly.

Solution

I think the graphs are the easiest thing to interpret, but use whatever you got:

- The females on average were younger than the males. (This was still true with the medians, even though those very old males might have pulled the mean up.)
- The people who survived were on average younger than those who didn't (or, the older people tended not to survive).
- A greater proportion of females survived than males.

A relevant point about each relationship is good.

(d) Fit a logistic regression predicting survival from age and gender. Display the summary.

Solution

Each row of the data frame is one person, so we can use the `survived` column without going through that two-column response business. However, the response variable `survived` is a categorical variable expressed as text, so we need to make it into a `factor` first. Either create a new variable that is the factor version of `survived`, or do it right in the `glm`:

```r
donner.1 = glm(factor(survived) ~ age + gender,
    family = "binomial", data = donner)
summary(donner.1)
```

```
##
## Call:
## glm(formula = factor(survived) ~ age + gender, family = "binomial",
##     data = donner)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -1.7445  -1.0441  -0.3029   0.8877   2.0472
##
## Coefficients:
##             Estimate Std. Error z value
## (Intercept)  3.23041    1.38686   2.329
## age         -0.07820    0.03728  -2.097
## gendermale  -1.59729    0.75547  -2.114
##             Pr(>|z|)
## (Intercept)   0.0198 *
## age           0.0359 *
## gendermale    0.0345 *
## ---
## Signif. codes:
##   0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1  ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 61.827  on 44  degrees of freedom
## Residual deviance: 51.256  on 42  degrees of freedom
## AIC: 57.256
##
## Number of Fisher Scoring iterations: 4
```

We ought to take a moment to think about what is being predicted here:

```r
levels(factor(donner$survived))
```

```
## [1] "no"  "yes"
```

The baseline is the first of these, `no`, and the thing that is predicted is the probability of the second one, `yes` (that is, the probability of surviving).

(e) Do both explanatory variables have an impact on survival? Does that seem to be consistent with your numerical or graphical summaries? Explain briefly.

Solution

Both explanatory variables have a P-value (just) less than 0.05, so they both have an impact on survival: taking either one of them out of the logistic regression would be a mistake. To see if this makes sense, go back to your plots or summaries, the ones involving survival. For age, the mean or median age of the survivors was less than for the people who died, by five year (median) or eight years (mean), so it makes sense that there would be an age effect. For gender, two-thirds of the women survived and two-thirds of the men died, so there ought to be a gender effect and is.

(f) Are the men typically older, younger or about the same age as the women? Considering this, explain carefully what the negative `gendermale` slope in your logistic regression means.

Solution

The men are typically older than the women. The negative (and significant) `gendermale` slope means that the probability of a male surviving is less than that of a woman *of the same age*. Or, though the males are typically older, *even after you allow for that*, the males have worse survival. (`genderfemale` is the baseline.) You need to get at the idea of "all else equal" when you're assessing regression slopes of any kind: regular regression, logistic regression or survival analysis (coming up later). That's why I said "carefully" in the question. If I say "carefully" or "precisely", a complete answer is looking for a specific thing to show that you understand the issue completely.

(g) Obtain the median and quartiles of age, and construct a data frame that contains all combinations of those ages and the two genders, with columns called exactly `age` and `gender`. Use this data frame to obtain predicted probabilities of survival for each of those combinations of age and gender.

Solution

I have a standard way of doing this:

- Make a list of all the variable values I'm going to predict for, using plural names
- Make a data frame containing all combinations, using `crossing`, giving the columns the right names
- Feed this data frame into `predict`, along with a fitted model, and get a column of predictions.
- Glue those predictions onto the data frame of values to predict for, and display them.

That way I can do it without thinking too hard. There is a lazy way to do step 1 (which I'll show first), and a way that is more work but more reliable (because you have to concentrate less). Step 1, the lazy way:

```
donner %>% select(age) %>% summary()
```

```
##       age
##  Min.   :15.0
##  1st Qu.:24.0
##  Median :28.0
##  Mean   :31.8
##  3rd Qu.:40.0
##  Max.   :65.0
```

```
donner %>% count(gender)
```

```
## # A tibble: 2 x 2
##   gender     n
##   <chr>  <int>
## 1 female    15
## 2 male      30
```

Then construct vectors called `ages` and `genders` by literally copying the values output from above:

```
ages = c(24, 28, 40)
genders = c("female", "male")
```

There is a risk here, because it is very easy to make typos while copying, so it would be more reliable to compute these (and, as long as you trust your computation, reproducible as well). Thinking of reproducibility, if the data set changes, your computation of quartiles etc. will still work, whereas your copying above won't.

So, let's get the median and quartiles of age:

```
donner %>% summarize(q1 = quantile(age, 0.25),
    med = median(age), q3 = quantile(age, 0.75))
```

```
## # A tibble: 1 x 3
##      q1   med    q3
##   <dbl> <int> <dbl>
## 1    24    28    40
```

only these came out sideways, so we can "transpose" them:

```
donner %>% summarize(q1 = quantile(age, 0.25),
    med = median(age), q3 = quantile(age, 0.75)) %>%
    gather(stat, value, everything())
```

```
## Warning: attributes are not identical across measure variables;
## they will be dropped
```

```
## # A tibble: 3 x 2
##   stat  value
##   <chr> <dbl>
## 1 q1       24
## 2 med      28
## 3 q3       40
```

and now if we "pull out" the `value` column, we are good:

```
ages = donner %>% summarize(q1 = quantile(age,
    0.25), med = median(age), q3 = quantile(age,
    0.75)) %>% gather(stat, value, everything()) %>%
    pull(value)
```

```
## Warning: attributes are not identical across measure variables;
## they will be dropped
```

```
ages
```

```
## [1] 24 28 40
```

The same kind of idea will get us both genders without typing any genders:

```
genders = donner %>% count(gender) %>% pull(gender)
genders
```

```
## [1] "female" "male"
```

We don't care how many there are of each gender; it's just a device to get the different genders. This is another way:

The counts are in a column called *n* which we calculate and then ignore.

```
genders = donner %>% select(gender) %>% distinct() %>%
    pull(gender)
genders
```

```
## [1] "male"   "female"
```

The rationale behind this one is that `distinct()` removes duplicates (or duplicate combinations if you have more than one variable), leaving you with only the distinct ones (the ones that are different from each other).

I don't think the genders are ever going to change, but it's the principle of the thing.

Step 2:

```
donner.new = crossing(age = ages, gender = genders)
donner.new
```

```
## # A tibble: 6 x 2
##     age gender
##   <dbl> <chr>
## 1    24 female
## 2    24 male
## 3    28 female
## 4    28 male
## 5    40 female
## 6    40 male
```

All combinations of the three ages and two genders.

Step 3:

```
p = predict(donner.1, donner.new, type = "response")
p
```

```
##         1         2         3         4
## 0.7947039 0.4393557 0.7389850 0.3643360
##         5         6
## 0.5255405 0.1831661
```

Step 4:

```
cbind(donner.new, p)
```

```
##   age gender         p
## 1  24 female 0.7947039
## 2  24   male 0.4393557
## 3  28 female 0.7389850
## 4  28   male 0.3643360
## 5  40 female 0.5255405
## 6  40   male 0.1831661
```

These, remember, are predicted probabilities of *surviving*.

(h) Do your predictions support your conclusions from earlier about the effects of age and gender? Explain briefly.

Solution

We said before that the probability of survival was lower if the age was higher. This is confirmed here: for example, look at rows 1, 3 and 5, which are all females of increasing ages; the probability of survival decreases. (Or look at males, in rows 2, 4 and 6; the effect is the same.) To see the effect of gender, look at two predictions of different

genders but the same age (eg. rows 1 and 2). The female is always predicted to have the higher survival probability. This is also what we saw before. The effect of gender is substantial, but not strongly significant, because we only have 45 observations, not so many when all we know about each person is whether they survived or not. I wanted you to think about these different ways to understand the model, and to understand that they all say the same thing, in different ways (and thus you can look at whichever of them is most convenient or comprehensible). For the logistic and survival models, I find looking at predictions to be the easiest way to understand what the model is saying.

16.9 *Who needs the most intensive care?*

The "APACHE II" is a scale for assessing patients who arrive in the intensive care unit (ICU) of a hospital. These are seriously ill patients who may die despite the ICU's best attempts. APACHE stands for "Acute Physiology And Chronic Health Evaluation". The scale score is calculated from several physiological measurements such as body temperature, heart rate and the Glasgow coma scale, as well as the patient's age. The final result is a score between 0 and 71, with a higher score indicating more severe health issues. Is it true that a patient with a higher APACHE II score has a higher probability of dying?

As with many of these acronyms, you get the idea that the acronym came first and they devised some words to fit it.

Data from one hospital are in link. The columns are: the APACHE II score, the total number of patients who had that score, and the number of patients with that score who died.

(a) Read in and display the data (however much of it displays). Why are you convinced that have the right thing?

Solution

Data values separated by one space, so:

```
my_url = "http://www.utsc.utoronto.ca/~butler/d29/apache.txt"
icu = read_delim(my_url, " ")
```

```
## Parsed with column specification:
## cols(
##   apache = col_integer(),
##   patients = col_integer(),
##   deaths = col_integer()
## )
```

```
icu
```

```
## # A tibble: 38 x 3
##    apache patients deaths
##     <int>    <int>  <int>
##  1      0        1      0
##  2      2        1      0
##  3      3        4      1
##  4      4       11      0
##  5      5        9      3
##  6      6       14      3
##  7      7       12      4
##  8      8       22      5
##  9      9       33      3
## 10     10       19      6
## # ... with 28 more rows
```

I had to stop and think about what to call the data frame, since one of the columns is called `apache`.

Anyway, I appear to have an `apache` score between 0 and something, a number of patients and a number of deaths (that is no bigger than the number of patients). If you check the original data, the `apache` scores go up to 41 and are all the values except for a few near the end, so it makes perfect sense that there would be 38 rows.

Basically, any comment here is good, as long as you make one and it has something to do with the data.

`apache` scores could be as high as 71, but I imagine a patient would have to be *very* ill to get a score anywhere near that high.

(b) Does each row of the data frame relate to one patient or sometimes to more than one? Explain briefly.

Solution

Sometimes to more than one. The number in the `patients` column says how many patients that line refers to: that is to say (for example) the row where `apache` equals 6 represents *all* the patients whose `apache` score was 6, however many of them there were (14 in this case). I had to be careful with the wording because the first two rows of the data frame actually *do* refer to only one patient each (who survived in both cases), but the later rows do refer to more than one patient.

(c) Explain why this is the kind of situation where you need a two-column response, and create this response variable, bearing in mind that I will (later) want you to estimate the probability of dying, given the `apache` score.

Solution

This needs a two-column response precisely *because* each row represents (or could represent) more than one observation. The two columns are the number of observations referring to the event of interest (dying), and the number of observations where that didn't happen (survived). We don't actually have the numbers of survivals, but we can calculate these by subtracting from the numbers of patients (since a patient must have either lived or died):

```
response = icu %>% mutate(survivals = patients -
    deaths) %>% select(deaths, survivals) %>%
    as.matrix()
response
```

```
##       deaths survivals
##  [1,]      0         1
##  [2,]      0         1
##  [3,]      1         3
##  [4,]      0        11
##  [5,]      3         6
##  [6,]      3        11
##  [7,]      4         8
##  [8,]      5        17
##  [9,]      3        30
## [10,]      6        13
## [11,]      5        26
## [12,]      5        12
## [13,]     13        19
## [14,]      7        18
## [15,]      7        11
## [16,]      8        16
## [17,]      8        19
## [18,]     13         6
## [19,]      7         8
## [20,]      6         7
## [21,]      9         8
## [22,]     12         2
## [23,]      7         6
## [24,]      8         3
## [25,]      8         4
## [26,]      2         4
## [27,]      5         2
## [28,]      1         2
## [29,]      4         3
## [30,]      4         1
## [31,]      3         0
```

```
## [32,]      3         0
## [33,]      1         0
## [34,]      1         0
## [35,]      1         0
## [36,]      1         0
## [37,]      1         0
## [38,]      0         1
```

noting that the deaths column has to come *first* since that's what we want the probability of. It has to be a `matrix`, so `as.matrix` is the final step. You can quickly check that the two numbers in each row add up to the number of `patients` for that row.

Or do everything outside of the data frame:

```
survivals = with(icu, patients - deaths)
resp = with(icu, cbind(deaths, survivals))
resp
```

```
##       deaths survivals
##  [1,]      0         1
##  [2,]      0         1
##  [3,]      1         3
##  [4,]      0        11
##  [5,]      3         6
##  [6,]      3        11
##  [7,]      4         8
##  [8,]      5        17
##  [9,]      3        30
## [10,]      6        13
## [11,]      5        26
## [12,]      5        12
## [13,]     13        19
## [14,]      7        18
## [15,]      7        11
## [16,]      8        16
## [17,]      8        19
## [18,]     13         6
## [19,]      7         8
## [20,]      6         7
## [21,]      9         8
## [22,]     12         2
## [23,]      7         6
## [24,]      8         3
## [25,]      8         4
## [26,]      2         4
```

```
## [27,]      5         2
## [28,]      1         2
## [29,]      4         3
## [30,]      4         1
## [31,]      3         0
## [32,]      3         0
## [33,]      1         0
## [34,]      1         0
## [35,]      1         0
## [36,]      1         0
## [37,]      1         0
## [38,]      0         1
```

```
class(resp)
```

```
## [1] "matrix"
```

Or use the dollar sign instead of the `withs`. Any of those is good.

I have no objection to your displaying the response matrix.

(d) Fit a logistic regression to estimate the probability of death from the `apache` score, and display the results.

Solution

```
apache.1 = glm(response ~ apache, family = "binomial",
    data = icu)
summary(apache.1)
```

```
##
## Call:
## glm(formula = response ~ apache, family = "binomial", data = icu)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -2.2508  -0.5662   0.1710   0.6649   2.3695
##
## Coefficients:
##             Estimate Std. Error z value
## (Intercept)  -2.2903     0.2765  -8.282
## apache        0.1156     0.0160   7.227
##             Pr(>|z|)
## (Intercept)  < 2e-16 ***
## apache      4.94e-13 ***
## ---
## Signif. codes:
```

```
##   0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1  ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 106.009  on 37  degrees of freedom
## Residual deviance:  43.999  on 36  degrees of freedom
## AIC: 125.87
##
## Number of Fisher Scoring iterations: 4
```

My naming convention has gotten messed up again. This should really be called `deaths.1` or something like that, but that would be a really depressing name.

(e) Is there a significant effect of `apache` score on the probability of survival? Explain briefly.

Solution

A gimme two points. The P-value for `apache` is 4.94×10^{-13}, very small, so `apache` score definitely has an effect on the probability of survival.

(f) Is the effect of a larger `apache` score to increase or to decrease the probability of death? Explain briefly.

Solution

The slope coefficient for `apache` is 0.1156, positive, and since we are modelling the probability of death (the first column of the response matrix), this says that as `apache` goes up, the probability of death goes up as well. If you made your response matrix with the columns the wrong way around, the slope coefficient for `apache` should be -0.1156, but the explanation should come to the same place, because this says that the probability of survival goes down as `apache` goes up.

(g) Obtain the predicted probability of death for each of the `apache` scores that were in the data set. Display these predicted probabilities next to the `apache` values that they came from. (You can display all of them.)

Solution

This is the easier version of `predict` since we don't have to make a new data frame of values to predict from:

```
p = predict(apache.1, type = "response")
cbind(icu, p)
```

```
##    apache patients deaths          p
## 1       0        1      0 0.09192721
## 2       2        1      0 0.11313881
## 3       3        4      1 0.12526979
## 4       4       11      0 0.13849835
## 5       5        9      3 0.15287970
## 6       6       14      3 0.16846238
## 7       7       12      4 0.18528595
## 8       8       22      5 0.20337870
## 9       9       33      3 0.22275512
## 10     10       19      6 0.24341350
## 11     11       31      5 0.26533372
## 12     12       17      5 0.28847528
## 13     13       32     13 0.31277589
## 14     14       25      7 0.33815068
## 15     15       18      7 0.36449223
## 16     16       24      8 0.39167143
## 17     17       27      8 0.41953937
## 18     18       19     13 0.44793012
## 19     19       15      7 0.47666442
## 20     20       13      6 0.50555404
## 21     21       17      9 0.53440661
## 22     22       14     12 0.56303077
## 23     23       13      7 0.59124118
## 24     24       11      8 0.61886322
## 25     25       12      8 0.64573711
## 26     26        6      2 0.67172126
## 27     27        7      5 0.69669475
## 28     28        3      1 0.72055875
## 29     29        7      4 0.74323712
## 30     30        5      4 0.76467607
## 31     31        3      3 0.78484314
## 32     32        3      3 0.80372553
## 33     33        1      1 0.82132803
## 34     34        1      1 0.83767072
## 35     35        1      1 0.85278652
## 36     36        1      1 0.86671872
## 37     37        1      1 0.87951866
## 38     41        1      0 0.92058988
```

The `type="response"` is needed to make sure the predictions come out as probabilities. If you omit it, you get log-odds.

The predicted probability (of dying) does indeed go up as `apache` goes up.

(h) Make a plot of predicted death probability against `apache` score (joined by lines) with, also on the plot, the observed proportion of deaths within each `apache` score, plotted against `apache` score. Does there seem to be good agreement between observation and prediction?

Solution

This means calculating the observed proportions first, adding the predicted probabilities, and then making the plot, like this:

```
icu %>% mutate(obs_prop = deaths/patients) %>%
    mutate(pred = p) %>% ggplot(aes(x = apache,
    y = pred)) + geom_line() + geom_point(aes(y = obs_prop))
```

You don't need to make a column in the pipeline with the predictions in it; you can just use what I called `p` directly in the `aes`.

Note that you *do* need to have a new `aes` inside the `geom_point`, however, because the y of the plot has changed: it needs to be the observed proportion now.

What you actually have to do in this situation depends on what you have. In this case, we have the total number of patients at each `apache` score, but you might have the number of patients surviving in one column and dying in another, in which case you'd need to calculate the total first.

I'd say the agreement is pretty good, except for the one patient with `apache` of 41 but who somehow survived.

That's what I asked for, and is full marks if you got it. However, the points are not all based on the same number of observations. One way to show that on your plot is to vary the size of the plotted point according to the number of patients it was based on. This is not hard to do, since we have exactly that in `patients`:

By size is meant the *area* of the circle, which is what our brains perceive as the size of two-dimensional, like the area of a slice in a pie chart. On the plot, the radius of the circle for 20 is less than twice that of the circle for 10, because the area depends on the radius *squared*.

```
icu %>% mutate(obs_prop = deaths/patients) %>%
    mutate(pred = p) %>% ggplot(aes(x = apache,
    y = pred)) + geom_line() + geom_point(aes(y = obs_prop,
    size = patients))
```

The points that are far from the prediction are mostly based on a small number of patients, and the observed proportions for `apache` scores with a lot of patients are mostly close to the prediction. Note that the `size`, because it is based on one of the variables, goes *inside* the `aes`. If you wanted the points all to be of size 5, say, you'd do it this way:

```
geom_point(aes(y = obs_prop), size = 5)
```

16.10 *Go away and don't come back!*

When a person has a heart attack and survives it, the major concern of health professionals is to prevent the person having a second heart attack. Two factors that are believed to be important are anger and anxiety; if a heart attack survivor tends to be angry or anxious, they are believed to put themselves at increased risk of a second heart attack.

Twenty heart attack survivors took part in a study. Each one was given a test designed to assess their anxiety (a higher score on the test indicates higher anxiety), and some of the survivors took an anger management course. The data are in link; `y` and `n` denote "yes" and "no" respectively. The columns denote (i) whether or not the person had a second heart attack, (ii) whether or not the person took the anger management class, (iii) the anxiety score.

(a) Read in and display the data.

Solution

```
my_url = "http://www.utsc.utoronto.ca/~butler/d29/ha2.txt"
ha = read_delim(my_url, " ")
```

```
## Parsed with column specification:
## cols(
##   second = col_character(),
##   anger = col_character(),
##   anxiety = col_integer()
## )
```

```
ha
```

```
## # A tibble: 20 x 3
##    second anger anxiety
##    <chr>  <chr>   <int>
##  1 y      y          70
##  2 y      y          80
##  3 y      y          50
##  4 y      n          60
##  5 y      n          40
##  6 y      n          65
##  7 y      n          75
##  8 y      n          80
##  9 y      n          70
## 10 y      n          60
## 11 n      y          65
## 12 n      y          50
## 13 n      y          45
## 14 n      y          35
## 15 n      y          40
## 16 n      y          50
## 17 n      n          55
## 18 n      n          45
## 19 n      n          50
## 20 n      n          60
```

The anxiety scores are numbers; the other two variables are "yes" and "no", which makes perfect sense.

(b)* Fit a logistic regression predicting whether or not a heart attack survivor has a second heart attack, as it depends on anxiety score and whether or not the person took the anger management class. Display the results.

Solution

This:

```r
ha.1 = glm(factor(second) ~ anger + anxiety, family = "binomial",
    data = ha)
summary(ha.1)
```

```
##
## Call:
## glm(formula = factor(second) ~ anger + anxiety, family = "binomial",
##     data = ha)
##
## Deviance Residuals:
##      Min        1Q    Median        3Q
## -1.52106  -0.68746   0.00424   0.70625
##      Max
##  1.88960
##
## Coefficients:
##             Estimate Std. Error z value
## (Intercept) -6.36347    3.21362  -1.980
## angery      -1.02411    1.17101  -0.875
## anxiety      0.11904    0.05497   2.165
##             Pr(>|z|)
## (Intercept)   0.0477 *
## angery        0.3818
## anxiety       0.0304 *
## ---
## Signif. codes:
##   0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1  ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 27.726  on 19  degrees of freedom
## Residual deviance: 18.820  on 17  degrees of freedom
## AIC: 24.82
##
## Number of Fisher Scoring iterations: 4
```

There's no need to worry about two-column responses here, because each row of the data frame refers to only one person, and the response variable `second` is already a categorical variable with two categories (that needs to be turned into a `factor` for `glm`).

(c) In the previous part, how can you tell that you were predicting the probability of having a second heart attack (as opposed to the probability of not having one)?

Solution

The levels of a factor are taken in alphabetical order, with n as the baseline, so we are predicting the probability of the second one y.

(d)* For the two possible values y and n of anger and the anxiety scores 40, 50 and 60, make a data frame containing all six combinations, and use it to obtain predicted probabilities of a second heart attack. Display your predicted probabilities side by side with what they are predictions for.

Solution

This time, I give you the values I want the predictions for (as opposed to calculating something like quartiles from the data), so we might as well just type them in. Step 1 is to save them (my preference is under plural names):

```
anxieties = c(40, 50, 60)
angers = c("y", "n")
```

Step 2 is to make a data frame of combinations using crossing. I'll call mine new:

```
new = crossing(anxiety = anxieties, anger = angers)
new
```

```
## # A tibble: 6 x 2
##   anxiety anger
##     <dbl> <chr>
## 1      40 n
## 2      40 y
## 3      50 n
## 4      50 y
## 5      60 n
## 6      60 y
```

Step 3 is to do the predictions. Into predict go (in order) the fitted model, the new data frame that we just made, and something to make the predictions be probabilities:

```
p = predict(ha.1, new, type = "response")
```

Step 4 is to put these next to the data frame we created. cbind is easiest, since we don't know what kind of thing p might be, and bind_cols is pickier:

```
cbind(new, p)
```

```
##   anxiety anger          p
## 1      40     n 0.16774789
## 2      40     y 0.06749763
```

```
## 3      50     n 0.39861864
## 4      50     y 0.19226955
## 5      60     n 0.68551307
## 6      60     y 0.43908386
```

(e) Use your predictions from the previous part to describe the effect of changes in `anxiety` and `anger` on the probability of a second heart attack.

Solution

The idea is to change one variable *while leaving the other fixed*. (This is the common refrain of "all else equal".) Pick a level of `anger`, say `n` (it doesn't matter which) and look at the effect of `anxiety`. The probability of a second heart attack increases sharply from 0.17 to 0.40 to 0.69. So an increased anxiety score is associated with an increased probability of second heart attack (all else equal). To assess the effect of taking the anger management course, pick an `anxiety` value, say 40, and compare the probabilities for `anger` `n` and `y`. For someone who has not taken the anger management course, the probability is 0.17, but for someone who has, it drops all the way to 0.07. (The pattern is the same at the other anxiety scores.) Extra: the reason it doesn't matter what value of the other variable you look at (as long as you keep it fixed) is that the model is "additive" with no interaction, so that the effect of one variable does not depend on the value of the other one. If we wanted to see whether the effect of anxiety was different according to whether or not the person had taken the anger management course, we would add an interaction between `anxiety` and `anger`. But we won't be doing this kind of thing until we get to analysis of variance, so you don't need to worry about it yet.

(f) Are the effects you described in the previous part consistent with the `summary` output from `glm` that you obtained in (here)? Explain briefly how they are, or are not. (You need an explanation for each of `anxiety` and `anger`, and you will probably get confused if you look at the P-values, so don't.)

Solution

In the previous part, we found that increased anxiety went with an increased probability of second heart attack. Back in (here), we found a positive slope of 0.11904 for anxiety, which also means that a higher anxiety score goes with a higher probability of second heart attack. That was not too hard. The other one is a little more work. `anger` is categorical with two categories `n` and `y`. The first one, `n`, is the baseline, so `angery` shows how `y` compares to `n`. The slope -1.04211 is negative, which means that someone who has taken the anger management course has a *lower* probability of a second heart attack

than someone who hasn't (all else equal). This is the same story that we got from the predictions. That's it for that, but I suppose I should talk about those P-values. `anxiety` is significant, so it definitely has an effect on the probability of a second heart attack. The pattern is clear enough even with this small data set. Here's a visual:

```
ggplot(ha, aes(x = second, y = anxiety)) + geom_boxplot()
```

This is the wrong way around in terms of cause and effect, but it shows pretty clearly that people who suffer a second heart attack have a higher level of anxiety than those who don't.

That's clear enough, but what about `anger`? That is *not* significant, but there seems to be a visible effect of `anger` on the predicted probabilities of (here): there we saw that if you had done the anger management course, your probability of a second heart attack was lower. But that's only the predicted probability, and there is also uncertainty about that, probably quite a lot because we don't have much data. So if we were to think about confidence intervals for the predicted probabilities, they would be wide, and for the two levels of `anger` at a fixed `anxiety` they would almost certainly overlap.

Another way of seeing that is a visual, which would be a side-by-side bar chart:

```
ggplot(ha, aes(x = anger, fill = second)) + geom_bar(position = "dodge")
```

A small majority of people who took the anger management did not have a second heart attack, while a small minority of those who did not, did. But with these small numbers, the difference, even though it points the way we would have guessed, is not large enough to be significant:

Read that carefully.

```
with(ha, table(anger, second))
```

```
##      second
## anger n y
##     n 4 7
##     y 6 3
```

This is not nearly far enough from an equal split to be significant. (It's the same kind of idea as for Mood's median test in C32.)

17
Logistic regression with ordinal or nominal response

```
library(MASS)
library(nnet)
library(tidyverse)
```

```
## -- Attaching packages ---- tidyverse 1.2.1 --
```

```
## v ggplot2 3.1.0     v purrr   0.2.5
## v tibble  1.4.2     v dplyr   0.7.8
## v tidyr   0.8.1     v stringr 1.3.1
## v readr   1.1.1     v forcats 0.3.0
```

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

17.1 *Do you like your mobile phone?*

A phone company commissioned a survey of their customers' satisfaction with their mobile devices. The responses to the survey were on a so-called Likert scale of "very unsatisfied", "unsatisfied", "satisfied", "very satisfied". Also recorded were each customer's gender and age group (under 18, 18–24, 25–30, 31 or older). (A survey of this kind does not ask its respondents for their exact age, only which age group they fall in.) The data, as frequencies of people falling into each category combination, are in link.

(a) *Read in the data and take a look at the format. Use a tool that you know about to arrange the frequencies in* one* column, with other columns labelling the response categories that the frequencies belong to. Save the new data frame. (Take a look at it if you like.)

Solution

```
my_url = "http://www.utsc.utoronto.ca/~butler/d29/mobile.txt"
mobile = read_delim(my_url, " ")
```

```
## Parsed with column specification:
## cols(
##   gender = col_character(),
##   age.group = col_character(),
##   very.unsat = col_integer(),
##   unsat = col_integer(),
##   sat = col_integer(),
##   very.sat = col_integer()
## )
```

```
mobile
```

```
## # A tibble: 8 x 6
##   gender age.group very.unsat unsat   sat
##   <chr>  <chr>          <int> <int> <int>
## 1 male   0-17               3     9    18
## 2 male   18-24              6    13    16
## 3 male   25-30              9    13    17
## 4 male   31+                5     7    16
## 5 female 0-17               4     8    11
## 6 female 18-24              8    14    20
## 7 female 25-30             10    15    16
## 8 female 31+                5    14    12
## # ... with 1 more variable: very.sat <int>
```

With multiple columns that are all frequencies, this is a job for `gather`:

```
mobile.long = mobile %>% gather(satisfied, frequency,
    very.unsat:very.sat)
mobile.long
```

```
## # A tibble: 32 x 4
##    gender age.group satisfied  frequency
##    <chr>  <chr>     <chr>          <int>
##  1 male   0-17      very.unsat         3
##  2 male   18-24     very.unsat         6
##  3 male   25-30     very.unsat         9
##  4 male   31+       very.unsat         5
##  5 female 0-17      very.unsat         4
##  6 female 18-24     very.unsat         8
##  7 female 25-30     very.unsat        10
##  8 female 31+       very.unsat         5
##  9 male   0-17      unsat              9
## 10 male   18-24     unsat             13
## # ... with 22 more rows
```

Yep, all good. See how `mobile.long` contains what it should? (For those keeping track, the original data frame had 8 rows and 4 columns to gather, and the new one has $8 \times 4 = 32$ rows.)

(b) We are going to fit ordered logistic models below. To do that, we need our response variable to be a factor with its levels in the right order. By looking at the data frame you just created, determine what kind of thing your intended response variable currently is.

Solution

I looked at `mobile.long` in the previous part, but if you didn't, look at it here:

```
mobile.long
```

```
## # A tibble: 32 x 4
##    gender age.group satisfied  frequency
##    <chr>  <chr>     <chr>          <int>
##  1 male   0-17      very.unsat         3
##  2 male   18-24     very.unsat         6
##  3 male   25-30     very.unsat         9
##  4 male   31+       very.unsat         5
##  5 female 0-17      very.unsat         4
##  6 female 18-24     very.unsat         8
##  7 female 25-30     very.unsat        10
##  8 female 31+       very.unsat         5
##  9 male   0-17      unsat              9
## 10 male   18-24     unsat             13
## # ... with 22 more rows
```

My intended response variable is what I called `satisfied`. This is `chr` or "text", not the `factor` that I want.

(c) If your intended response variable is not a factor, create a factor in your data frame with levels in the right order. Hint: look at the order your levels are *in the data*.

Solution

My intended response `satisfied` is text, not a factor, so I need to do this part. The hint is to look at the column `satisfied` in `mobile.long` and note that the satisfaction categories appear in the data *in the order that we want*. This is good news, because we can use `fct_inorder` like this:

```
mobile.long <- mobile.long %>% mutate(satis = fct_inorder(satisfied))
```

If you check, by looking at the data frame, `satis` is a `factor`, and you can also do this to verify that its levels are in the right order:

```
with(mobile.long, levels(satis))
```

```
## [1] "very.unsat" "unsat"      "sat"
## [4] "very.sat"
```

Success.

Extra: so now you are asking, what if the levels are in the *wrong* order in the data? Well, below is what you used to have to do, and it will work for this as well. I'll first find what levels of satisfaction I have. This can be done by counting them, or by finding the distinct ones:

```
mobile.long %>% count(satisfied)
```

```
## # A tibble: 4 x 2
##   satisfied      n
##   <chr>      <int>
## 1 sat            8
## 2 unsat          8
## 3 very.sat       8
## 4 very.unsat     8
```

or

```
mobile.long %>% distinct(satisfied)
```

```
## # A tibble: 4 x 1
##   satisfied
##   <chr>
## 1 very.unsat
## 2 unsat
## 3 sat
## 4 very.sat
```

If you count them, they come out in alphabetical order. If you ask for the distinct ones, they come out in the order they were in `mobile.long`, which is the order the *columns* of those names were in `mobile`, which is the order we want.

To actually grab those satisfaction levels as a vector (that we will need in a minute), use `pluck` to pull the column out of the data frame as a vector:

```
v1 = mobile.long %>% distinct(satisfied) %>% pluck("satisfied")
v1
```

```
## [1] "very.unsat" "unsat"      "sat"
## [4] "very.sat"
```

which is in the correct order, or

```
v2 = mobile.long %>% count(satisfied) %>% pluck("satisfied")
v2
```

```
## [1] "sat"        "unsat"      "very.sat"
## [4] "very.unsat"
```

which is in alphabetical order. The problem with the second one is that we know the correct order, but there isn't a good way to code that, so we have to rearrange it ourselves. The correct order from `v2` is 4, 2, 1, 3, so:

```
v3 = c(v2[4], v2[2], v2[1], v2[3])
v3
```

```
## [1] "very.unsat" "unsat"      "sat"
## [4] "very.sat"
```

```
v4 = v2[c(4, 2, 1, 3)]
v4
```

```
## [1] "very.unsat" "unsat"      "sat"
## [4] "very.sat"
```

Either of these will work. The first one is more typing, but is perhaps more obvious. There is a third way, which is to keep things as a data frame until the end, and use `slice` to pick out the rows in the right order:

```
v5 = mobile.long %>% count(satisfied) %>% slice(c(4,
    2, 1, 3)) %>% pluck("satisfied")
v5
```

```
## [1] "very.unsat" "unsat"      "sat"
## [4] "very.sat"
```

If you don't see how that works, run it yourself, one line at a time.

The other way of doing this is to physically type them into a vector, but this carries the usual warnings of requiring you to be very careful and that it won't be reproducible (eg. if you do another survey with different response categories).

So now create the proper response variable thus, using your vector of categories:

```
mobile.long2 <- mobile.long %>% mutate(satis = ordered(satisfied,
    v1))
mobile.long2
```

```
## # A tibble: 32 x 5
##    gender age.group satisfied frequency sat~
##    <chr>  <chr>     <chr>         <int> <or>
##  1 male   0-17      very.uns~         3 ver~
##  2 male   18-24     very.uns~         6 ver~
##  3 male   25-30     very.uns~         9 ver~
##  4 male   31+       very.uns~         5 ver~
##  5 female 0-17      very.uns~         4 ver~
##  6 female 18-24     very.uns~         8 ver~
##  7 female 25-30     very.uns~        10 ver~
##  8 female 31+       very.uns~         5 ver~
##  9 male   0-17      unsat             9 uns~
## 10 male   18-24     unsat            13 uns~
## # ... with 22 more rows
```

`satis` has the same values as `satisfied`, but its label `ord` means that it is an ordered factor, as we want.

(d)* Fit ordered logistic models to predict satisfaction from (i) gender and age group, (ii) gender only, (iii) age group only. (You don't need to examine the models.) Don't forget a suitable `weights`!

Solution

(i):

```
library(MASS)
mobile.1 = polr(satis ~ gender + age.group, weights = frequency,
    data = mobile.long)
```

For (ii) and (iii), `update` is the thing (it works for any kind of model):

```
mobile.2 = update(mobile.1, . ~ . - age.group)
mobile.3 = update(mobile.1, . ~ . - gender)
```

We're not going to look at these, because the output from `summary` is not very illuminating. What we do next is to try to figure out which (if either) of the explanatory variables `age.group` and `gender` we need.

(e) Use `drop1` on your model containing both explanatory variables to determine whether you can remove either of them. Use `test="Chisq"` to obtain P-values.

Solution

`drop1` takes a fitted model, and tests each term in it in turn, and says which (if any) should be removed. Here's how it goes:

```
drop1(mobile.1, test = "Chisq")
```

```
## Single term deletions
##
## Model:
## satis ~ gender + age.group
##           Df    AIC     LRT Pr(>Chi)
## <none>       1101.8
## gender     1 1104.2  4.4089 0.035751 *
## age.group  3 1109.0 13.1641 0.004295 **
## ---
## Signif. codes:
##   0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1  ' ' 1
```

The possibilities are to remove `gender`, to remove `age.group` or to remove nothing. The best one is "remove nothing", because it's the one on the output with the smallest AIC. Both P-values are small, so it would be a mistake to remove either of the explanatory variables.

(f) Use `anova` to decide whether we are justified in removing `gender` from a model containing both `gender` and `age.group`. Compare your P-value with the one from `drop1`.

Solution

This is a comparison of the model with both variables (`mobile.1`) and the model with `gender` removed (`mobile.3`). Use `anova` for this, smaller (fewer-x) model first:

```
anova(mobile.3, mobile.1)
```

```
## Likelihood ratio tests of ordinal regression models
##
## Response: satis
##                Model Resid. df Resid. Dev
## 1          age.group       414   1092.213
## 2 gender + age.group       413   1087.804
##     Test    Df LR stat.    Pr(Chi)
## 1
## 2 1 vs 2     1  4.40892 0.03575146
```

The P-value is (just) less than 0.05, so the models are significantly different. That means that the model with both variables in fits significantly better than the model with only `age.group`, and therefore that taking `gender` out is a mistake.

The P-value is identical to the one from `drop1` (because they are both doing the same test).

(g) Use `anova` to see whether we are justified in removing `age.group` from a model containing both `gender` and `age.group`. Compare your P-value with the one from `drop1` above.

Solution

Exactly the same idea as the last part. In my case, I'm comparing models `mobile.2` and `mobile.1`:

```
anova(mobile.2, mobile.1)
```

```
## Likelihood ratio tests of ordinal regression models
##
## Response: satis
##                Model Resid. df Resid. Dev
## 1             gender       416   1100.968
## 2 gender + age.group       413   1087.804
##     Test    Df LR stat.     Pr(Chi)
## 1
## 2 1 vs 2     3 13.16411 0.004294811
```

This one is definitely significant, so I need to keep `age.group` for sure. Again, the P-value is the same as the one in `drop1`.

(h) Which of the models you have fit so far is the most appropriate one? Explain briefly.

Solution

I can't drop either of my variables, so I have to keep them both: `mobile.1`, with both `age.group` and `gender`.

(i) Obtain predicted probabilities of a customer falling in the various satisfaction categories, as it depends on gender and age group. To do that, you need to feed `predict` three things: the fitted model that contains both age group and gender, the data frame that you read in from the file back in part (here) (which contains all the combinations of age group and gender), and an appropriate `type`.

Solution

My model containing both xs was `mobile.1`, the data frame read in from the file was called `mobile`, and I need `type="p"` to get probabilities:

```
probs = predict(mobile.1, mobile, type = "p")
mobile %>% select(gender, age.group) %>% cbind(probs)
```

```
##   gender age.group very.unsat     unsat
## 1   male      0-17 0.06070852 0.1420302
## 2   male     18-24 0.09212869 0.1932086
## 3   male     25-30 0.13341018 0.2438110
## 4   male       31+ 0.11667551 0.2252966
## 5 female      0-17 0.08590744 0.1840411
```

```
## 6 female     18-24 0.12858414 0.2387296
## 7 female     25-30 0.18290971 0.2853880
## 8 female       31+ 0.16112036 0.2692996
##         sat  very.sat
## 1 0.2752896 0.5219716
## 2 0.3044732 0.4101894
## 3 0.3084506 0.3143282
## 4 0.3097920 0.3482359
## 5 0.3011751 0.4288763
## 6 0.3091489 0.3235374
## 7 0.2920053 0.2396970
## 8 0.3008711 0.2687089
```

This worked for me, but this might happen to you, with the same commands as above:

```
## Error in MASS::select(., gender, age.group): unused arguments (gender, age.group)
```

Oh, this didn't work. Why not? There don't seem to be any errors.

This is the kind of thing that can bother you for *days*. The resolution (that it took me a long time to discover) is that you might have the `tidyverse` *and also* `MASS` loaded, in the wrong order, and `MASS` also has a `select` (that takes different inputs and does something different). If you look back at part (here), you might have seen a message there when you loaded `MASS` that `select` was "masked". When you have two packages that both contain a function with the same name, the one that you can see (and that will get used) is the one that was loaded *last*, which is the `MASS` select (not the one we actually wanted, which is the `tidyverse` select). There are a couple of ways around this. One is to un-load the package we no longer need (when we no longer need it). The mechanism for this is shown at the end of part (here). The other is to say explicitly which package you want your function to come from, so that there is no doubt. The `tidyverse` is actually a collection of packages. The best way to find out which one our `select` comes from is to go to the Console window in R Studio and ask for the help for `select`. With both `tidyverse` and `MASS` loaded, the help window offers you a choice of both `select`s; the one we want is "select/rename variables by name", and the actual package it comes from is `dplyr`.

With that in mind, we can get the right `select` like this:

```
probs = predict(mobile.1, mobile, type = "p")
mobile %>% dplyr::select(gender, age.group) %>%
    cbind(probs)
```

```
##   gender age.group very.unsat     unsat
```

```
## 1   male      0-17 0.06070852 0.1420302
## 2   male     18-24 0.09212869 0.1932086
## 3   male     25-30 0.13341018 0.2438110
## 4   male       31+ 0.11667551 0.2252966
## 5 female      0-17 0.08590744 0.1840411
## 6 female     18-24 0.12858414 0.2387296
## 7 female     25-30 0.18290971 0.2853880
## 8 female       31+ 0.16112036 0.2692996
##         sat  very.sat
## 1 0.2752896 0.5219716
## 2 0.3044732 0.4101894
## 3 0.3084506 0.3143282
## 4 0.3097920 0.3482359
## 5 0.3011751 0.4288763
## 6 0.3091489 0.3235374
## 7 0.2920053 0.2396970
## 8 0.3008711 0.2687089
```

This gets the right `select` for sure: package name, two colons, function name means "the function by that name that lives in that package".

This is a non-standard use of `cbind` because I wanted to grab only the gender and age group columns from `mobile` first, and then `cbind` *that* to the predicted probabilities. The missing first input to `cbind` is "whatever came out of the previous step", that is, the first two columns of `mobile`.

I only included the first two columns of `mobile` in the `cbind`, because the rest of the columns of `mobile` were frequencies, which I don't need to see. (Having said that, it would be interesting to make a *plot* using the observed proportions and predicted probabilities, but I didn't ask you for that.)

This is an unusual `predict`, because we didn't have to make a data frame (with my usual name `new`) containing all the combinations of things to predict for. We were lucky enough to have those already in the original data frame `mobile`.

The usual way to do this is something like the trick that we did for getting the different satisfaction levels:

```
genders = mobile.long %>% distinct(gender) %>%
    pluck("gender")
age.groups = mobile.long %>% distinct(age.group) %>%
    pluck("age.group")
```

This is getting perilously close to deserving a function written to do it (strictly speaking, we should, since this is three times we've used this idea now).

Then `crossing` to get the combinations, and then `predict`:

```
new = crossing(gender = genders, age.group = age.groups)
new
```

```
## # A tibble: 8 x 2
##   gender age.group
##   <chr>  <chr>
## 1 female 0-17
## 2 female 18-24
## 3 female 25-30
## 4 female 31+
## 5 male   0-17
## 6 male   18-24
## 7 male   25-30
## 8 male   31+
```

```
pp = predict(mobile.1, new, type = "p")
cbind(new, pp)
```

```
##   gender age.group very.unsat     unsat
## 1 female      0-17 0.08590744 0.1840411
## 2 female     18-24 0.12858414 0.2387296
## 3 female     25-30 0.18290971 0.2853880
## 4 female       31+ 0.16112036 0.2692996
## 5   male      0-17 0.06070852 0.1420302
## 6   male     18-24 0.09212869 0.1932086
## 7   male     25-30 0.13341018 0.2438110
## 8   male       31+ 0.11667551 0.2252966
##         sat  very.sat
## 1 0.3011751 0.4288763
## 2 0.3091489 0.3235374
## 3 0.2920053 0.2396970
## 4 0.3008711 0.2687089
## 5 0.2752896 0.5219716
## 6 0.3044732 0.4101894
## 7 0.3084506 0.3143282
## 8 0.3097920 0.3482359
```

This method is fine if you want to do this question this way; the way I suggested first ought to be easier, but the nice thing about this is its mindlessness: you always do it about the same way.

(j)* Describe any patterns you see in the predictions, bearing in mind the significance or not of the explanatory variables.

Solution

I had both explanatory variables being significant, so I would expect to see both an age-group effect *and* a gender effect. For both males and females, there seems to be a decrease in satisfaction as the customers get older, at least until age 30 or so. I can see this because the predicted prob. of "very satisfied" decreases, and the predicted prob. of "very unsatisfied" increases. The 31+ age group are very similar to the 25–30 group for both males and females. So that's the age group effect. What about a gender effect? Well, for all the age groups, the males are more likely to be very satisfied than the females of the corresponding age group, and also less likely to to be very unsatisfied. So the gender effect is that males are more satisfied than females overall. (Or, the males are less discerning. Take your pick.) When we did the tests above, age group was very definitely significant, and gender less so (P-value around 0.03). This suggests that the effect of age group ought to be large, and the effect of gender not so large. This is about what we observed: the age group effect was pretty clear, and the gender effect was noticeable but small: the females were less satisfied than the males, but there wasn't all that much difference. Now we need to "unload" MASS, because, as we saw, *it* has a `select` that we don't want to have interfering with the `dplyr` one:

```
detach("package:MASS", unload = T)
```

17.2 *Attitudes towards abortion*

* Abortion is a divisive issue in the United States, particularly among Christians, some of whom believe that abortion is absolutely forbidden by the Bible. Do attitudes towards abortion differ among Christian denominations? The data in link come from the American Social Survey for the years 1972–1974. The variables are:

- `year`: year of survey, 1972, 1973 or 1974
- `religion`: Christian denomination: Southern Protestant, other Protestant, Catholic.
- `education`: years of education: low (0–8), medium (8–12), high (more than 12).
- `attitude` towards abortion (response). There were three abortion questions in the survey, asking whether the respondent thought that abortion should be legal in these three circumstances:
- when there is a strong possibility of a birth defect
- when the mother's health is threatened

- when the pregnancy is the result of rape.

A respondent who responded "yes" to all three questions was recorded as having a Positive attitude towards abortion; someone who responded "no" to all three was recorded as Negative, and anyone who gave a mixture of Yes and No responses was recorded as Mixed.

- `frequency` The number of respondents falling into the given category combination.

(a) Read in and display the data. Do you seem to have the right things?

Solution

The columns are aligned with each other but not with the headings, so `read_table2` it has to be:

```
my_url = "http://www.utsc.utoronto.ca/~butler/d29/abortion.txt"
abortion = read_table2(my_url)
```

```
## Parsed with column specification:
## cols(
##   year = col_integer(),
##   religion = col_character(),
##   education = col_character(),
##   attitude = col_character(),
##   frequency = col_integer()
## )
```

```
abortion
```

```
## # A tibble: 81 x 5
##     year religion education attitude
##    <int> <chr>    <chr>     <chr>
##  1  1972 Prot     Low       Neg
##  2  1972 Prot     Low       Mix
##  3  1972 Prot     Low       Pos
##  4  1972 Prot     Med       Neg
##  5  1972 Prot     Med       Mix
##  6  1972 Prot     Med       Pos
##  7  1972 Prot     High      Neg
##  8  1972 Prot     High      Mix
##  9  1972 Prot     High      Pos
## 10  1972 SProt    Low       Neg
## # ... with 71 more rows, and 1 more
## #   variable: frequency <int>
```

This looks good. `religion, education, attitude` are correctly categorical with apparently the right levels, and `frequency` is a (whole) number.

The only weirdness is that year got read in as a number, because it *was* a number. `read_table2` had no way of knowing that we might have wanted to treat this as categorical. But you don't need to observe that.

(b) We have three choices for fitting logistic regressions:

- `glm` with `family="binomial"`
- `polr` (from package `MASS`)
- `multinom` (from package `nnet`)

Why should we be using `polr` for this analysis? Explain briefly.

Solution

It all comes down to what kind of response variable we have. Our response variable `attitude` has *three* categories that can be put in order: negative, mixed, positive (or the other way around). So we have an ordinal response, which calls for `polr`. You can eliminate the other possibilities, because `glm` requires a *two*-category response and `multinom` requires a nominal response whose categories do *not* have a natural order. In short, "because we have an ordered (ordinal) response".

(c) `attitude` is text, and needs to be an ordered factor, with the values in the right order. Create a vector, using `c()` if you wish, that contains the factor levels in the right order, and then create a new `attitude` factor using the `ordered` function, feeding it first the variable as read from the file, and second the new ordered levels that you want it to have.

Solution

This is actually harder to describe than it is to do. First thing is to decide on the ordering you want. You can go low to high or high to low (it doesn't matter). I'm going negative to positive. Either way, you want `Mix` in the middle:

The results might look different which way you go, but it won't make any *material* difference.

```
lev = c("Neg", "Mix", "Pos")
```

Then use the original `attitude` plus this `lev` as input to `ordered`:

```
abortion = abortion %>% mutate(attitude.ord = ordered(attitude,
    lev))
abortion
```

```
## # A tibble: 81 x 6
##     year religion education attitude
##    <int> <chr>    <chr>     <chr>
##  1  1972 Prot     Low       Neg
##  2  1972 Prot     Low       Mix
##  3  1972 Prot     Low       Pos
##  4  1972 Prot     Med       Neg
##  5  1972 Prot     Med       Mix
##  6  1972 Prot     Med       Pos
##  7  1972 Prot     High      Neg
##  8  1972 Prot     High      Mix
##  9  1972 Prot     High      Pos
## 10  1972 SProt    Low       Neg
## # ... with 71 more rows, and 2 more
## #   variables: frequency <int>,
## #   attitude.ord <ord>
```

I printed out the result to convince myself that I had the right thing, but you don't have to. Note that `attitude.ord` has the same values as `attitude`, but is `ord` (that is, an ordered factor) rather than text.

This is the easy way. To save yourself some typing, you can get the `attitude` levels from the data:

```
lev = abortion %>% distinct(attitude) %>% pull(attitude)
lev
```

```
## [1] "Neg" "Mix" "Pos"
```

or

```
lev2 = abortion %>% count(attitude) %>% pull(attitude)
lev2
```

```
## [1] "Mix" "Neg" "Pos"
```

`distinct` arranges the values in the order it found them in the data, which is a sensible low-to-high here; `count` arranges the levels in alphabetical order, which is *not* sensible, so you'd need to rearrange them, like this:

```
lev3 = lev2[c(2, 1, 3)]
lev3
```

```
## [1] "Neg" "Mix" "Pos"
```

or, if you like working with data frames better:

```
lev4 = abortion %>% count(attitude) %>% slice(c(2,
    1, 3)) %>% pull(attitude)
lev4
```

```
## [1] "Neg" "Mix" "Pos"
```

where you'd have to go as far as `count` the first time to find what to slice, and then run the whole thing the second time.

(d) Fit a model that will predict the correctly-ordered attitude towards abortion from religious denomination and education. Don't forget that each row of the data file encodes a lot of people. You don't need to display the results.

Solution

This is a veiled hint to remember the `weights=` thing. Don't forget to use your `attitude` in the proper order that you went to such great pains to make in the last part!

```
library(MASS)
```

```
##
## Attaching package: 'MASS'

## The following object is masked from 'package:dplyr':
##
##     select
```

```
abortion.1 = polr(attitude.ord ~ religion + education,
    data = abortion, weights = frequency)
```

(e) Fit two more models, one containing `religion` only, and the other containing `education` only. Again, no need to look at the results.

Solution

This is a standard application of `update`. Note that this carries along the `data=`, and *also* the `weights=`, so you don't need either of them again; just make sure you say what *changes*, and everything else (including the form of the model) will stay the same:

```
abortion.2 = update(abortion.1, . ~ . - education)
abortion.3 = update(abortion.1, . ~ . - religion)
```

The syntax means "take everything in the model abortion.1, and then take out education" for the first one. If you don't like `update`, you can also copy and paste and edit:

```
abortion.2a = polr(attitude.ord ~ religion, data = abortion,
    weights = frequency)
abortion.3a = polr(attitude.ord ~ education, data = abortion,
    weights = frequency)
```

(f) Investigate whether each of `religion` and `education` should stay in the model, or whether they can be removed. Do this two ways, using `drop1` and using `anova`. What do you conclude?

Solution

I think `drop1` is easier, so let's do that first:

```
drop1(abortion.1, test = "Chisq")
```

```
## Single term deletions
##
## Model:
## attitude.ord ~ religion + education
##           Df    AIC    LRT  Pr(>Chi)
## <none>       4077.5
## religion   2 4139.2 65.635 5.592e-15 ***
## education  2 4152.6 79.104 < 2.2e-16 ***
## ---
## Signif. codes:
##   0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1  ' ' 1
```

Dropping nothing is rather clearly the best thing to do. Note how easy that is, because you test both variables at once. If one of them had been removable, you would have removed it, and then done `drop1` again to see whether the other one could come out as well.

The `anova` way is more complicated, but sometimes that's the only way that works. The idea is that you compare the fit of the model containing both variables with the model obtained by dropping one of the variables, and see whether you really need the bigger model. This has to be done twice, once for each explanatory variable you're testing.

First, we test `religion`, by comparing the model with it (and `education`) and the model without it (with just `education`, but the other way around (since the smaller model goes first):

```
anova(abortion.3, abortion.1)
```

```
## Likelihood ratio tests of ordinal regression models
##
## Response: attitude.ord
##                  Model Resid. df Resid. Dev
## 1            education      3233   4131.176
## 2 religion + education      3231   4065.541
##     Test    Df LR stat.      Pr(Chi)
## 1
## 2 1 vs 2     2 65.63505 5.551115e-15
```

This P-value is small, so the bigger model is better: `religion` should stay. (To be sure that you did the right thing, look in the `anova` output at the two `Model` lines: the extra thing in the bigger model needs to be the thing you're testing for.)

Now, we test `education` by comparing the model with both and the model without `education` (but with `religion` since that's important):

```
anova(abortion.2, abortion.1)
```

```
## Likelihood ratio tests of ordinal regression models
##
## Response: attitude.ord
##                  Model Resid. df Resid. Dev
## 1             religion      3233   4144.645
## 2 religion + education      3231   4065.541
##     Test    Df LR stat. Pr(Chi)
## 1
## 2 1 vs 2     2 79.10415       0
```

Same conclusion: `education` has to stay too. Note that the P-values in `drop1` and in `anova`, and also the test statistics (`LRT` and `LR stat.`) are the same, so it literally doesn't matter which way you do it.

(g) Which of your fitted models is the best? Explain briefly.

Solution

We couldn't take either of the explanatory variables out, so the model with both variables is best, the one I called `abortion.1`.

(h)* Obtain predicted probabilities for each attitude towards abortion for each combination of education level and religious denomination. To do that, (i) obtain those levels of education and religion, (ii) make a data frame out of all combinations of them, (iii) feed your best fitted model and that data frame into `predict` with any necessary options.

Solution

Use either `distinct` or `count` to get the levels of `education` and `religion`:

```
educations = abortion %>% distinct(education) %>%
    pull(education)
religions = abortion %>% distinct(religion) %>%
    pull(religion)
new = crossing(education = educations, religion = religions)
new
```

```
## # A tibble: 9 x 2
##   education religion
##   <chr>     <chr>
## 1 High      Cath
## 2 High      Prot
## 3 High      SProt
## 4 Low       Cath
## 5 Low       Prot
## 6 Low       SProt
## 7 Med       Cath
## 8 Med       Prot
## 9 Med       SProt
```

```
pp = predict(abortion.1, new, type = "p")
cbind(new, pp)
```

```
##   education religion        Neg        Mix
## 1      High     Cath 0.05681385 0.16191372
## 2      High     Prot 0.02649659 0.08579955
## 3      High    SProt 0.03342471 0.10504280
## 4       Low     Cath 0.15709933 0.30706616
## 5       Low     Prot 0.07767439 0.20363294
## 6       Low    SProt 0.09665550 0.23547512
## 7       Med     Cath 0.08336876 0.21375231
## 8       Med     Prot 0.03947421 0.12089996
## 9       Med    SProt 0.04962265 0.14566284
##         Pos
## 1 0.7812724
## 2 0.8877039
## 3 0.8615325
## 4 0.5358345
## 5 0.7186927
## 6 0.6678694
## 7 0.7028789
## 8 0.8396258
## 9 0.8047145
```

(i) Using your predictions from (here), describe the effect of education. (This should be the same for all values of `religion`).

Solution

Pick a religious denomination, say `Cath` since that's the first one. Focus on the three rows with that `religion`, here rows 1, 4 and 7. I'd say the striking thing is that people with `Low` education are most likely to have a negative attitude towards abortion. Or you could say that they are *least* likely to have a positive attitude towards abortion.

That's equally good. Or you can turn it around and say that the `High` education people are *more* likely to be in favour of abortion (or less likely to be opposed to it). This holds true for all religious denominations, as you can check. This is one of those cases where I don't mind much what you say, as long as the data (here the predictions) support it.

(j) Using your predictions from (here), describe the effect of `religion`. (This should be the same for all levels of `education`.)

Solution

As in the previous part: pick a level of `education`, and then judge the effect of `religion`. The `Low` level of education appeared to be the most opposed to abortion before, so the effects of `religion` might be the most noticeable there. For those, the Catholics are most likely to be negatively inclined towards abortion, and the least likely to be postively inclined. There is not much difference between the two flavours of Protestant. Some further thoughts: What actually surprises me is that you hear (in the media) about Christians being steadfastly opposed to abortion. This is something that these data do not support at all: an appreciable fraction of people from each of the denominations, especially with medium or high levels of education, are actually in favour of abortion in all three of the circumstances described in the question. The models we fitted assume an effect of `religion` regardless of education, and an effect of `education` regardless of religion. But it might be that the effect of education depends on which religious denomination you're looking at. The way to assess whether that is true is to add an *interaction*, as you would in analysis of variance. (We haven't talked about that in this course yet, which is why I didn't ask you to do it.) In R, `a:b` means "the interaction between factors `a` and `b`" and `a*b` means "the main effects of `a` and `b` and their interaction". In our case, therefore, we should add `education:religion` to our model and test it for significance:

```
abortion.4 = update(abortion.1, . ~ . + education:religion)
anova(abortion.1, abortion.4)
```

```
## Likelihood ratio tests of ordinal regression models
##
## Response: attitude.ord
##                                        Model
## 1                       religion + education
## 2 religion + education + religion:education
##   Resid. df Resid. Dev   Test    Df LR stat.
## 1      3231   4065.541
## 2      3227   4047.041 1 vs 2     4 18.49958
```

```
##        Pr(Chi)
## 1
## 2 0.0009853301
```

The interaction *is* significant, so it should stay. What effect does it have on the predictions? We've done all the setup earlier, so we just run `predict` again on the new model:

```
pp = predict(abortion.4, new, type = "p")
cbind(new, pp)
```

```
##   education religion        Neg        Mix
## 1      High     Cath 0.07624521 0.20160124
## 2      High     Prot 0.02248256 0.07434749
## 3      High    SProt 0.02363341 0.07775848
## 4       Low     Cath 0.11358676 0.26036670
## 5       Low     Prot 0.08192969 0.21185155
## 6       Low    SProt 0.12342416 0.27283576
## 7       Med     Cath 0.08153700 0.21115988
## 8       Med     Prot 0.04087702 0.12486250
## 9       Med    SProt 0.04858716 0.14369236
##         Pos
## 1 0.7221536
## 2 0.9031699
## 3 0.8986081
## 4 0.6260465
## 5 0.7062188
## 6 0.6037401
## 7 0.7073031
## 8 0.8342605
## 9 0.8077205
```

For Southern Protestants and Other Protestants, the effect of education is as we described it before: the lower the amount of education, the less favourably inclined towards abortion (that is, `Neg` is more likely and `Pos` is less likely). This is still true for Catholics, but the effect of education is less noticeable: the probabilities of each response, over the levels of education, vary much less. This, I think, is where the interaction comes from: for Catholics, education has a much smaller effect on attitudes towards abortion than it does for either of the Protestant denominations.

I would have liked to have you explore this, but the question was already too long, so I called a halt where I did.

Again, we should tidy up after ourselves:

```
detach("package:MASS", unload = T)
```

17.3 *Finding non-missing values*

* This is to prepare you for something in the next question. It's meant to be easy.

In R, the code `NA` stands for "missing value" or "value not known". In R, `NA` should not have quotes around it. (It is a special code, not a piece of text.)

(a) Create a vector `v` that contains some numbers and some missing values, using `c()`. Put those values into a one-column data frame.

Solution

Like this. The arrangement of numbers and missing values doesn't matter, as long as you have some of each:

```
v = c(1, 2, NA, 4, 5, 6, 9, NA, 11)
mydata = tibble(v)
mydata
```

```
## # A tibble: 9 x 1
##       v
##   <dbl>
## 1     1
## 2     2
## 3    NA
## 4     4
## 5     5
## 6     6
## 7     9
## 8    NA
## 9    11
```

This has one column called `v`.

(b) Obtain a new column containing `is.na(v)`. When is this true and when is this false?

Solution

```
mydata = mydata %>% mutate(isna = is.na(v))
mydata
```

```
## # A tibble: 9 x 2
##       v isna
##   <dbl> <lgl>
## 1     1 FALSE
## 2     2 FALSE
```

```
## 3    NA TRUE
## 4     4 FALSE
## 5     5 FALSE
## 6     6 FALSE
## 7     9 FALSE
## 8    NA TRUE
## 9    11 FALSE
```

This is `TRUE` if the corresponding element of `v` is missing (in my case, the third value and the second-last one), and `FALSE` otherwise (when there is an actual value there).

(c) The symbol `!` means "not" in R (and other programming languages). What does `!is.na(v)` do? Create a new column containing that.

Solution

Try it and see. Give it whatever name you like. My name reflects that I know what it's going to do:

```
mydata = mydata %>% mutate(notisna = !is.na(v))
mydata
```

```
## # A tibble: 9 x 3
##       v isna  notisna
##   <dbl> <lgl> <lgl>
## 1     1 FALSE TRUE
## 2     2 FALSE TRUE
## 3    NA TRUE  FALSE
## 4     4 FALSE TRUE
## 5     5 FALSE TRUE
## 6     6 FALSE TRUE
## 7     9 FALSE TRUE
## 8    NA TRUE  FALSE
## 9    11 FALSE TRUE
```

This is the logical opposite of `is.na`: it's true if there is a value, and false if it's missing.

(d) Use `filter` to display just the rows of your data frame that have a non-missing value of `v`.

Solution

`filter` takes a column to say which rows to pick, in which case the column should contain something that either *is* `TRUE` or `FALSE`, or something that can be interpreted that way:

```
mydata %>% filter(notisna)
```

```
## # A tibble: 7 x 3
##       v isna  notisna
##   <dbl> <lgl> <lgl>
## 1     1 FALSE TRUE
## 2     2 FALSE TRUE
## 3     4 FALSE TRUE
## 4     5 FALSE TRUE
## 5     6 FALSE TRUE
## 6     9 FALSE TRUE
## 7    11 FALSE TRUE
```

or you can provide `filter` something that can be calculated from what's in the data frame, and also returns something that is either true or false:

```
mydata %>% filter(!is.na(v))
```

```
## # A tibble: 7 x 3
##       v isna  notisna
##   <dbl> <lgl> <lgl>
## 1     1 FALSE TRUE
## 2     2 FALSE TRUE
## 3     4 FALSE TRUE
## 4     5 FALSE TRUE
## 5     6 FALSE TRUE
## 6     9 FALSE TRUE
## 7    11 FALSE TRUE
```

In either case, I only have non-missing values of `v`.

17.4 *European Social Survey and voting*

The European Social Survey is a giant survey carried out across Europe covering demographic information, attitudes to and amount of education, politics and so on. In this question, we will investigate what might make British people vote for a certain political party.

The information for this question is in a (large) spreadsheet at link. There is also a "codebook" at link that tells you what all the variables are. The ones we will use are the last five columns of the spreadsheet, described on pages 11 onwards of the codebook. (I could have given you more, but I didn't want to make this any more complicated than it already was.)

(a) Read in the `.csv` file, and verify that you have lots of rows and columns.

Solution

The obvious way. Printing it out will display some of the data and tell you how many rows and columns you have:

```
my_url = "http://www.utsc.utoronto.ca/~butler/d29/ess.csv"
ess = read_csv(my_url)
```

```
## Parsed with column specification:
## cols(
##   cntry = col_character(),
##   cname = col_character(),
##   cedition = col_integer(),
##   cproddat = col_character(),
##   cseqno = col_integer(),
##   name = col_character(),
##   essround = col_integer(),
##   edition = col_double(),
##   idno = col_integer(),
##   dweight = col_double(),
##   pspwght = col_double(),
##   pweight = col_double(),
##   prtvtgb = col_integer(),
##   gndr = col_integer(),
##   agea = col_integer(),
##   eduyrs = col_integer(),
##   inwtm = col_integer()
## )
```

```
ess
```

```
## # A tibble: 2,286 x 17
##    cntry cname cedition cproddat cseqno name
##    <chr> <chr>    <int> <chr>     <int> <chr>
##  1 GB    ESS1~        1 26.11.2~ 134168 ESS6~
##  2 GB    ESS1~        1 26.11.2~ 134169 ESS6~
##  3 GB    ESS1~        1 26.11.2~ 134170 ESS6~
##  4 GB    ESS1~        1 26.11.2~ 134171 ESS6~
##  5 GB    ESS1~        1 26.11.2~ 134172 ESS6~
##  6 GB    ESS1~        1 26.11.2~ 134173 ESS6~
##  7 GB    ESS1~        1 26.11.2~ 134174 ESS6~
##  8 GB    ESS1~        1 26.11.2~ 134175 ESS6~
##  9 GB    ESS1~        1 26.11.2~ 134176 ESS6~
## 10 GB    ESS1~        1 26.11.2~ 134177 ESS6~
## # ... with 2,276 more rows, and 11 more
## #   variables: essround <int>,
## #   edition <dbl>, idno <int>,
```

```
## #   dweight <dbl>, pspwght <dbl>,
## #   pweight <dbl>, prtvtgb <int>,
## #   gndr <int>, agea <int>, eduyrs <int>,
## #   inwtm <int>
```

2286 rows and 17 columns.

(b)* Use the codebook to find out what the columns `prtvtgb`, `gndr`, `agea`, `eduyrs` and `inwtm` are. What do the values 1 and 2 for `gndr` mean? (You don't, at this point, have to worry about the values for the other variables.)

Solution

Respectively, political party voted for at last election, gender (of respondent), age at interview, years of full-time education, length of interview (in minutes). For `gndr`, male is 1 and female is 2.

(c) The three major political parties in Britain are the Conservative, Labour and Liberal Democrat. (These, for your information, correspond roughly to the Canadian Progressive Conservative, NDP and Liberal parties.) For the variable that corresponds to "political party voted for at the last election", which values correspond to these three parties?

Solution

1, 2 and 3 respectively. (That was easy!)

(d) Normally, I would give you a tidied-up data set. But I figure you could use some practice tidying this one up. As the codebook shows, there are some numerical codes for missing values, and we want to omit those. We want just the columns `prtvtgb` through `inwtm` from the right side of the spreadsheet. Use `dplyr` or `tidyr` tools to (i) select only these columns, (ii) include the rows that correspond to people who voted for one of the three major parties, (iii) include the rows that have an age at interview less than 999, (iv) include the rows that have less than 40 years of education, (v) include the rows that are not missing on `inwtm` (use the idea from Question~here for (v)). The last four of those (the inclusion of rows) can be done in one go.

Solution

This seems to call for a chain. The major parties are numbered 1, 2 and 3, so we can select the ones less than 4 (or `<=3`). The reference back to the last question is a hint to use `!is.na()`.

```
ess.major = ess %>% select(prtvtgb:inwtm) %>%
    filter(prtvtgb < 4, agea < 999, eduyrs < 40,
        !is.na(inwtm))
```

You might get a weird error in `select`, something about "unused argument". If this happens to you, it's *not* because you used `select` wrong, it's because you used the wrong `select`! There is one in `MASS`, and you need to make sure that this package is "detached" so that you use the `select` you want, namely the one in `dplyr`, loaded with the `tidyverse`. Use the instructions at the end of the mobile phones question or the abortion question to do this.

The other way around this is to say, instead of `select`, `dplyr::select` with two colons. This means "the `select` that lives in `dplyr`, no other", and is what Wikipedia calls "disambiguation": out of several things with the same name, you say which one you mean.

If you do the pipeline, you will probably *not* get it right the first time. (I didn't.) For debugging, try out one step at a time, and summarize what you have so far, so that you can check it for correctness. A handy trick for that is to make the last piece of your pipeline `summary()`, which produces a summary of the columns of the resulting data frame. For example, I first did this (note that my `filter` is a lot simpler than the one above):

```
ess %>% select(prtvtgb:inwtm) %>% filter(prtvtgb <
    4, !is.na(inwtm)) %>% summary()
```

```
##     prtvtgb           gndr
##  Min.   :1.000   Min.   :1.000
##  1st Qu.:1.000   1st Qu.:1.000
##  Median :2.000   Median :2.000
##  Mean   :1.803   Mean   :1.572
##  3rd Qu.:2.000   3rd Qu.:2.000
##  Max.   :3.000   Max.   :2.000
##       agea            eduyrs
##  Min.   : 18.00   Min.   : 0.00
##  1st Qu.: 44.00   1st Qu.:11.00
##  Median : 58.00   Median :13.00
##  Mean   : 61.74   Mean   :14.23
##  3rd Qu.: 71.00   3rd Qu.:16.00
##  Max.   :999.00   Max.   :88.00
##      inwtm
##  Min.   :  7.00
##  1st Qu.: 35.00
##  Median : 41.00
##  Mean   : 43.54
##  3rd Qu.: 50.00
##  Max.   :160.00
```

The mean of a categorical variable like party voted for or gender doesn't make much sense, but it looks as if all the values are sensible

ones (1 to 3 and 1, 2 respectively). However, the maximum values of age and years of education look like missing value codes, hence the other requirements I put in the question.

If you do not take out the *NA* values, they are shown separately on the end of the *summary* for that column.

`Printing` as the last step of your pipeline also works, but the advantage of `summary` is that you get to see whether there are any unusual values, in this case unusually *large* values that are missing value codes.

(e) Why is my response variable nominal rather than ordinal? How can I tell? Which R function should I use, therefore, to fit my model?

Solution

The response variable is political party voted for. There is no (obvious) ordering to this (unless you want to try to place the parties on a left-right spectrum), so this is nominal, and you'll need `multinom` from package `nnet`. If I had included the minor parties and you were working on a left-right spectrum, you would have had to decide where to put the somewhat libertarian Greens or the parties that exist only in Northern Ireland.

(f)* Take the political party voted for, and turn it into a factor, by feeding it into `factor`. Fit an appropriate model to predict political party voted for at the last election (as a factor) from all the other variables. Gender is really a categorical variable too, but since there are only two possible values it can be treated as a number.

Solution

This, or something like it. `multinom` lives in package `nnet`, which you'll have to install first if you haven't already:

```
library(nnet)
ess.1 = multinom(factor(prtvtgb) ~ gndr + agea +
    eduyrs + inwtm, data = ess.major)
```

```
## # weights:  18 (10 variable)
## initial  value 1343.602829
## iter  10 value 1256.123798
## final  value 1247.110080
## converged
```

Or create a factor version of your response in the data frame first:

```
ess.major = ess.major %>% mutate(party = factor(prtvtgb))
```

and then:

```
ess.1a = multinom(party ~ gndr + agea + eduyrs +
    inwtm, data = ess.major)
```

```
## # weights:  18 (10 variable)
## initial  value 1343.602829
## iter  10 value 1256.123798
## final  value 1247.110080
## converged
```

(g) We have a lot of explanatory variables. The standard way to test whether we need all of them is to take one of them out at a time, and test which ones we can remove. This is a lot of work. We won't do that. Instead, the R function `step` does what you want. You feed `step` two things: a fitted model object, and the option `trace=0` (otherwise you get a lot of output). The final part of the output from `step` tells you which explanatory variables you need to keep.

Run `step` on your fitted model. Which explanatory variables need to stay in the model here?

Solution

I tried to give you lots of hints here:

```r
step(ess.1, trace = 0)
```

```
## trying - gndr
## trying - agea
## trying - eduyrs
## trying - inwtm
## # weights:  15 (8 variable)
## initial  value 1343.602829
## iter  10 value 1248.343563
## final  value 1248.253638
## converged
## trying - agea
## trying - eduyrs
## trying - inwtm
```

```
## Call:
## multinom(formula = factor(prtvtgb) ~ agea + eduyrs + inwtm, data = ess.major)
##
## Coefficients:
##   (Intercept)        agea     eduyrs
## 2    1.632266 -0.02153694 -0.0593757
## 3   -1.281031 -0.01869263  0.0886487
##         inwtm
## 2 0.009615167
## 3 0.009337084
##
```

```
## Residual Deviance: 2496.507
## AIC: 2512.507
```

The end of the output gives us coefficients for (and thus tells us we need to keep) age, years of education and interview length.

The actual numbers don't mean much; it's the indication that the variable has stayed in the model that makes a difference.

There are three political parties; using the first as a baseline, there are therefore $3 - 1 = 2$ coefficients for each variable.

If you're wondering about the process: first `step` tries to take out each explanatory variable, one at a time, from the starting model (the one that contains all the variables). Then it finds the best model out of those and fits it. (It doesn't tell us which model this is, but evidently it's the one without gender.) Then it takes *that* model and tries to remove its explanatory variables one at a time (there are only three of them left). Having decided it cannot remove any of them, it stops, and shows us what's left.

Leaving out the `trace=0` shows more output and more detail on the process, but I figured this was enough (and this way, you don't have to wade through all of that output). Try values like 1 or 2 for `trace` and see what you get.

(h) Fit the model indicated by `step` (in the last part).

Solution

Copy and paste, and take out the variables you don't need. I found that gender needed to be removed, but if yours is different, follow through with whatever your `step` said to do.

```
ess.2 = multinom(party ~ agea + eduyrs + inwtm,
    data = ess.major)
```

```
## # weights:  15 (8 variable)
## initial  value 1343.602829
## iter  10 value 1248.343563
## final  value 1248.253638
## converged
```

(i) I didn't think that interview length could possibly be relevant to which party a person voted for. Test whether interview length can be removed from your model of the last part. What do you conclude? (Note that `step` and this test may disagree.)

Solution

Fit the model without `inwtm`:

```
ess.3 = multinom(party ~ agea + eduyrs, data = ess.major)
```

```
## # weights:  12 (6 variable)
```

```
## initial  value 1343.602829
## iter  10 value 1250.418281
## final  value 1250.417597
## converged
```

and then use `anova` to compare them:

```
anova(ess.3, ess.2)
```

```
## Likelihood ratio tests of Multinomial Models
##
## Response: party
##                   Model Resid. df Resid. Dev
## 1         agea + eduyrs      2440   2500.835
## 2 agea + eduyrs + inwtm      2438   2496.507
##     Test    Df LR stat.   Pr(Chi)
## 1
## 2 1 vs 2     2 4.327917 0.1148695
```

The P-value, 0.1149, is not small, which says that the smaller model is good, ie. the one *without* interview length.

I thought `drop1` would also work here, but it appears not to:

```
drop1(ess.1, test = "Chisq")
```

```
## trying - gndr
```

```
## Error in if (trace) {: argument is not interpretable as logical
```

I think that's a bug in `multinom`, since normally if `step` works, then `drop1` will work too (normally `step` *uses* `drop1`).

The reason for the disagreement between `step` and `anova` is that `step` will tend to keep marginal explanatory variables, that is, ones that are "potentially interesting" but whose P-values might not be less than 0.05. There is still no substitute for your judgement in figuring out what to do! `step` uses a thing called AIC to decide what to do, rather than actually doing a test. If you know about "adjusted R-squared" in choosing explanatory variables for a regression, it's the same idea: a variable can be not quite significant but still make the adjusted R-squared go up (typically only a little).

(j) Use your best model to obtain predictions from some suitably chosen combinations of values of the explanatory variables that remain. (If you have quantitative explanatory variables left, you could use their first and third quartiles as values to predict from. Running `summary` on the data frame will get summaries of all the variables.)

Solution

First make our new data frame of values to predict from. `crossing` is our friend. You can use `quantile` or `summary` to find the quartiles. I only had `agea` and `eduyrs` left, having decided that interview time really ought to come out:

```
summary(ess.major)
```

```
##     prtvtgb           gndr
##  Min.   :1.000   Min.   :1.000
##  1st Qu.:1.000   1st Qu.:1.000
##  Median :2.000   Median :2.000
##  Mean   :1.803   Mean   :1.574
##  3rd Qu.:2.000   3rd Qu.:2.000
##  Max.   :3.000   Max.   :2.000
##       agea           eduyrs
##  Min.   :18.00   Min.   : 0.00
##  1st Qu.:44.00   1st Qu.:11.00
##  Median :58.00   Median :13.00
##  Mean   :57.19   Mean   :13.45
##  3rd Qu.:71.00   3rd Qu.:16.00
##  Max.   :94.00   Max.   :33.00
##      inwtm       party
##  Min.   :  7.0   1:484
##  1st Qu.: 35.0   2:496
##  Median : 41.0   3:243
##  Mean   : 43.7
##  3rd Qu.: 50.0
##  Max.   :160.0
```

Quartiles for age are 44 and 71, and for years of education are 11 and 16:

```
new = crossing(agea = c(44, 71), eduyrs = c(11,
    16))
new
```

```
## # A tibble: 4 x 2
##    agea eduyrs
##   <dbl>  <dbl>
## 1    44     11
## 2    44     16
## 3    71     11
## 4    71     16
```

Now we feed this into `predict`. An annoying feature of this kind of prediction is that `type` may not be what you expect. The best model is the one I called `ess.3`:

```
pp = predict(ess.3, new, type = "probs")
cbind(new, pp)
```

```
##   agea eduyrs         1         2         3
## 1   44     11 0.3331882 0.5093898 0.1574220
## 2   44     16 0.3471967 0.3955666 0.2572367
## 3   71     11 0.4551429 0.4085120 0.1363451
## 4   71     16 0.4675901 0.3127561 0.2196538
```

(k) What is the effect of increasing age? What is the effect of an increase in years of education?

Solution

To assess the effect of age, hold years of education constant. Thus, compare lines 1 and 3 (or 2 and 4): increasing age tends to increase the chance that a person will vote Conservative (party 1), and decrease the chance that a person will vote Labour (party 2). There doesn't seem to be much effect of age on the chance that a person will vote Liberal Democrat. To assess education, hold age constant, and thus compare rows 1 and 2 (or rows 3 and 4). This time, there isn't much effect on the chances of voting Conservative, but as education increases, the chance of voting Labour goes down, and the chance of voting Liberal Democrat goes up. A little history: back 150 or so years ago, Britain had two political parties, the Tories and the Whigs. The Tories became the Conservative party (and hence, in Britain and in Canada, the Conservatives are nicknamed Tories). The Whigs became Liberals. At about the same time as working people got to vote (not women, yet, but working men) the Labour Party came into existence. The Labour Party has always been affiliated with working people and trades unions, like the NDP here. But power has typically alternated between Conservative and Labour goverments, with the Liberals as a third party. In the 1980s a new party called the Social Democrats came onto the scene, but on realizing that they couldn't make much of a dent by themselves, they merged with the Liberals to form the Liberal Democrats, which became a slightly stronger third party. I was curious about what the effect of interview length would be. Presumably, the effect is small, but I have no idea which way it would be. To assess this, it's the same idea over again: create a new data frame with all combinations of `agea`, `eduyrs` *and* `inwtm`. I need the quartiles of interview time first:

It amuses me that Toronto's current (2018) mayor is politically a Tory.

```
quantile(ess.major$inwtm)
```

```
##   0%  25%  50%  75% 100%
##    7   35   41   50  160
```

and then

```
ess.new = crossing(agea = c(44, 71), eduyrs = c(11,
    16), inwtm = c(35, 50))
ess.new
```

```
## # A tibble: 8 x 3
##    agea eduyrs inwtm
##   <dbl>  <dbl> <dbl>
## 1    44     11    35
## 2    44     11    50
## 3    44     16    35
## 4    44     16    50
## 5    71     11    35
## 6    71     11    50
## 7    71     16    35
## 8    71     16    50
```

and then predict *using the model that contained interview time*:

```
pp = predict(ess.2, ess.new, type = "probs")
cbind(ess.new, pp)
```

```
##   agea eduyrs inwtm         1         2
## 1   44     11    35 0.3455998 0.4993563
## 2   44     11    50 0.3139582 0.5240179
## 3   44     16    35 0.3606728 0.3872735
## 4   44     16    50 0.3284883 0.4074380
## 5   71     11    35 0.4810901 0.3886175
## 6   71     11    50 0.4455030 0.4157036
## 7   71     16    35 0.4945171 0.2968551
## 8   71     16    50 0.4589823 0.3182705
##           3
## 1 0.1550439
## 2 0.1620238
## 3 0.2520536
## 4 0.2640737
## 5 0.1302924
## 6 0.1387934
## 7 0.2086277
## 8 0.2227472
```

The effects of age and education are as they were before. A longer interview time is associated with a slightly decreased chance of voting Conservative and a slightly increased chance of voting Labour. Compare, for example, lines 1 and 2. But, as we suspected, the effect is small and not really worth worrying about.

17.5 *Alligator food*

What do alligators most like to eat? 219 alligators were captured in four Florida lakes. Each alligator's stomach contents were observed, and the food that the alligator had eaten was classified into one of five categories: fish, invertebrates (such as snails or insects), reptiles (such as turtles), birds, and "other" (such as amphibians, plants or rocks). The researcher noted for each alligator what that alligator had most of in its stomach, as well as the gender of each alligator and whether it was "large" or "small" (greater or less than 2.3 metres in length). The data can be found in link. The numbers in the data set (apart from the first column) are all frequencies. (You can ignore that first column "profile".)

Our aim is to predict food type from the other variables.

(a) Read in the data and display the first few lines. Describe how the data are not "tidy".

Solution

Separated by exactly one space:

```
my_url = "http://www.utsc.utoronto.ca/~butler/d29/alligator.txt"
gators.orig = read_delim(my_url, " ")
```

```
## Parsed with column specification:
## cols(
##   profile = col_integer(),
##   Gender = col_character(),
##   Size = col_character(),
##   Lake = col_character(),
##   Fish = col_integer(),
##   Invertebrate = col_integer(),
##   Reptile = col_integer(),
##   Bird = col_integer(),
##   Other = col_integer()
## )
```

```
gators.orig
```

```
## # A tibble: 16 x 9
##    profile Gender Size  Lake   Fish
##      <int> <chr>  <chr> <chr> <int>
##  1       1 f      <2.3  geor~     3
##  2       2 m      <2.3  geor~    13
##  3       3 f      >2.3  geor~     8
##  4       4 m      >2.3  geor~     9
```

```
##  5       5 f      <2.3  hanc~    16
##  6       6 m      <2.3  hanc~     7
##  7       7 f      >2.3  hanc~     3
##  8       8 m      >2.3  hanc~     4
##  9       9 f      <2.3  okla~     3
## 10      10 m      <2.3  okla~     2
## 11      11 f      >2.3  okla~     0
## 12      12 m      >2.3  okla~    13
## 13      13 f      <2.3  traf~     2
## 14      14 m      <2.3  traf~     3
## 15      15 f      >2.3  traf~     0
## 16      16 m      >2.3  traf~     8
## # ... with 4 more variables:
## #   Invertebrate <int>, Reptile <int>,
## #   Bird <int>, Other <int>
```

The last five columns are all frequencies. Or, one of the variables (food type) is spread over five columns instead of being contained in one. Either is good.

My choice of "temporary" name reflects that I'm going to obtain a "tidy" data frame called `gators` in a moment.

(b) Use `gather` to arrange the data suitably for analysis (which will be using `multinom`). Demonstrate (by looking at the first few rows of your new data frame) that you now have something tidy.

Solution

I'm creating my "official" data frame here:

```
gators = gators.orig %>% gather(Food.type, Frequency,
    Fish:Other)
gators
```

```
## # A tibble: 80 x 6
##    profile Gender Size  Lake  Food.type
##      <int> <chr>  <chr> <chr> <chr>
##  1       1 f      <2.3  geor~ Fish
##  2       2 m      <2.3  geor~ Fish
##  3       3 f      >2.3  geor~ Fish
##  4       4 m      >2.3  geor~ Fish
##  5       5 f      <2.3  hanc~ Fish
##  6       6 m      <2.3  hanc~ Fish
##  7       7 f      >2.3  hanc~ Fish
##  8       8 m      >2.3  hanc~ Fish
##  9       9 f      <2.3  okla~ Fish
## 10      10 m      <2.3  okla~ Fish
```

```
## # ... with 70 more rows, and 1 more
## #   variable: Frequency <int>
```

The usual thing: data frame, what makes the columns (to be gathered) different, what makes them the same, columns to gather. I gave my column names Capital Letters to make them consistent with the others (and in an attempt to stop infesting my brain with annoying variable-name errors when I fit models later).

Looking at the first few lines reveals that I now have a column of food types and one column of frequencies, both of which are what I wanted. I can check that I have all the different food types by finding the distinct ones:

```
gators %>% distinct(Food.type)
```

```
## # A tibble: 5 x 1
##   Food.type
##   <chr>
## 1 Fish
## 2 Invertebrate
## 3 Reptile
## 4 Bird
## 5 Other
```

(Think about why `count` would be confusing here.)

Note that `Food.type` is text (`chr`) rather than being a factor. I'll hold my breath and see what happens when I fit a model where it is supposed to be a factor.

(c) What is different about this problem, compared to Question~here, that would make `multinom` the right tool to use?

Solution

Look at the response variable `Food.type` (or whatever you called it): this has multiple categories, but they are *not ordered* in any logical way. Thus, in short, a nominal response.

(d) Fit a suitable multinomial model predicting food type from gender, size and lake. How you account for having `Frequency` as well?

Solution

The clue in the question is to remind you not to forget the `weights` piece, otherwise `multinom` will assume that you have *one* observation per line and not as many as the number in `Frequency`. *That* is the reason that `count` earlier would have been confusing: it would have told you how many *rows* contained each food type, rather than how many *alligators*, and these would have been different:

```
gators %>% count(Food.type)
```

```
## # A tibble: 5 x 2
##   Food.type        n
##   <chr>        <int>
## 1 Bird            16
## 2 Fish            16
## 3 Invertebrate    16
## 4 Other           16
## 5 Reptile         16
```

```
gators %>% count(Food.type, wt = Frequency)
```

```
## # A tibble: 5 x 2
##   Food.type        n
##   <chr>        <int>
## 1 Bird            13
## 2 Fish            94
## 3 Invertebrate    61
## 4 Other           32
## 5 Reptile         19
```

Each food type appears on 16 rows, but is the favoured diet of very different numbers of *alligators*. Note the use of `wt=` to specify a frequency variable.

Discovered by me two minutes ago.

You ought to understand *why* those are different.

All right, back to modelling:

```
library(nnet)
gators.1 = multinom(Food.type ~ Gender + Size +
    Lake, weights = Frequency, data = gators)
```

```
## # weights:  35 (24 variable)
## initial  value 352.466903
## iter  10 value 270.228588
## iter  20 value 268.944257
## final  value 268.932741
## converged
```

This worked, even though `Food.type` was actually text. I guess it got converted to a factor. The ordering of the levels doesn't matter here anyway, since this is not an ordinal model.

No need to look at it, since the output is kind of confusing anyway:

```
summary(gators.1)
```

```
## Call:
## multinom(formula = Food.type ~ Gender + Size + Lake, data = gators,
##     weights = Frequency)
##
## Coefficients:
##              (Intercept)     Genderm
## Fish           2.4322304  0.60674971
## Invertebrate   2.6012531  0.14378459
## Other          1.0014505  0.35423803
## Reptile       -0.9829064 -0.02053375
##                Size>2.3 Lakehancock
## Fish         -0.7308535  -0.5751295
## Invertebrate -2.0671545  -2.3557377
## Other        -1.0214847   0.1914537
## Reptile      -0.1741207   0.5534169
##              Lakeoklawaha Laketrafford
## Fish            0.5513785  -1.23681053
## Invertebrate    1.4645820  -0.08096493
## Other           0.5775317   0.32097943
## Reptile         3.0807416   1.82333205
##
## Std. Errors:
##              (Intercept)   Genderm  Size>2.3
## Fish           0.7706940 0.6888904 0.6523273
## Invertebrate   0.7917210 0.7292510 0.7084028
## Other          0.8747773 0.7623738 0.7250455
## Reptile        1.2827234 0.9088217 0.8555051
##              Lakehancock Lakeoklawaha
## Fish           0.7952147     1.210229
## Invertebrate   0.9463640     1.232835
## Other          0.9072182     1.374545
## Reptile        1.3797755     1.591542
##              Laketrafford
## Fish            0.8661187
## Invertebrate    0.8814625
## Other           0.9589807
## Reptile         1.3388017
##
## Residual Deviance: 537.8655
## AIC: 585.8655
```

You get one coefficient for each variable (along the top) and for each response group (down the side), using the first group as a baseline everywhere. These numbers are hard to interpret; doing predic-

tions is much easier.

(e) Do a test to see whether `Gender` should stay in the model. (This will entail fitting another model.) What do you conclude?

Solution

The other model to fit is the one *without* the variable you're testing:

```
gators.2 = update(gators.1, . ~ . - Gender)
```

```
## # weights:  30 (20 variable)
## initial  value 352.466903
## iter  10 value 272.246275
## iter  20 value 270.046891
## final  value 270.040139
## converged
```

I did `update` here to show you that it works, but of course there's no problem in just writing out the whole model again and taking out `Gender`, preferably by copying and pasting:

```
gators.2x = multinom(Food.type ~ Size + Lake,
    weights = Frequency, data = gators)
```

```
## # weights:  30 (20 variable)
## initial  value 352.466903
## iter  10 value 272.246275
## iter  20 value 270.046891
## final  value 270.040139
## converged
```

and then you compare the models with and without `Gender` using `anova`:

```
anova(gators.2, gators.1)
```

```
## Likelihood ratio tests of Multinomial Models
##
## Response: Food.type
##                  Model Resid. df Resid. Dev
## 1          Size + Lake       300   540.0803
## 2 Gender + Size + Lake       296   537.8655
##     Test    Df LR stat.   Pr(Chi)
## 1
## 2 1 vs 2     4 2.214796 0.6963214
```

The P-value is not small, so the two models fit equally well, and therefore we should go with the smaller, simpler one: that is, the one without `Gender`.

Sometimes `drop1` works here too (and sometimes it doesn't, for reasons I haven't figured out):

```
drop1(gators.1, test = "Chisq")
```

```
## trying - Gender
```

```
## Error in if (trace) {: argument is not interpretable as logical
```

I don't even know what this error message means, never mind what to do about it.

(f) Predict the probability that an alligator prefers each food type, given its size, gender (if necessary) and the lake it was found in, using the more appropriate of the two models that you have fitted so far. This means (i) obtaining all the sizes and lake names, (ii) making a data frame for prediction, and (iii) obtaining and displaying the predicted probabilities.

Solution

To get the different categories, use `distinct` and `pull`:

```
Lakes = gators %>% distinct(Lake) %>% pull(Lake)
Lakes
```

```
## [1] "george"   "hancock"  "oklawaha"
## [4] "trafford"
```

```
Sizes = gators %>% distinct(Size) %>% pull(Size)
Sizes
```

```
## [1] "<2.3" ">2.3"
```

I didn't need to think about `Genders` because that's not in the better model. See below for what happens if you include it anyway.

I have persisted with the Capital Letters, for consistency.

Next, a data frame for predicting from, using `crossing`, and called, as per my tradition, `new`:

```
new = crossing(Lake = Lakes, Size = Sizes)
new
```

```
## # A tibble: 8 x 2
##   Lake     Size
##   <chr>    <chr>
## 1 george   <2.3
## 2 george   >2.3
## 3 hancock  <2.3
```

```
## 4 hancock  >2.3
## 5 oklawaha <2.3
## 6 oklawaha >2.3
## 7 trafford <2.3
## 8 trafford >2.3
```

and then, obtain the predictions and glue them onto the data frame of values for which they are predictions. Don't forget to use the second model, the one without `Gender`. If you do forget, you'll get an error anyway, because your data frame of values to predict from doesn't have any `Gender` in it:

```
pp = predict(gators.2, new, type = "p")
preds1 = cbind(new, pp)
preds1
```

```
##       Lake Size        Bird      Fish
## 1   george <2.3 0.029671502 0.4521032
## 2   george >2.3 0.081071082 0.6574394
## 3  hancock <2.3 0.070400215 0.5353040
## 4  hancock >2.3 0.140898571 0.5701968
## 5 oklawaha <2.3 0.008818267 0.2581872
## 6 oklawaha >2.3 0.029419560 0.4584368
## 7 trafford <2.3 0.035892547 0.1842997
## 8 trafford >2.3 0.108222209 0.2957526
##   Invertebrate      Other    Reptile
## 1   0.41285699 0.09380190 0.01156641
## 2   0.13967877 0.09791193 0.02389880
## 3   0.09309885 0.25374163 0.04745531
## 4   0.02307179 0.19400899 0.07182382
## 5   0.60189518 0.05387241 0.07722691
## 6   0.24864408 0.06866206 0.19483754
## 7   0.51683770 0.17420330 0.08876673
## 8   0.19296148 0.20066241 0.20240133
```

Success. You won't get this right the first time. I certainly didn't. Anyway, these are the correct predictions that I discuss later.

If you thought that the better model was the one with `Gender` in it, or you otherwise forgot that you didn't need `Gender` then you needed to do something like this as well:

```
Genders = gators %>% distinct(Gender) %>% pull(Gender)
new = crossing(Lake = Lakes, Size = Sizes, Gender = Genders)
new
```

```
## # A tibble: 16 x 3
```

```
##    Lake     Size  Gender
##    <chr>    <chr> <chr>
##  1 george   <2.3  f
##  2 george   <2.3  m
##  3 george   >2.3  f
##  4 george   >2.3  m
##  5 hancock  <2.3  f
##  6 hancock  <2.3  m
##  7 hancock  >2.3  f
##  8 hancock  >2.3  m
##  9 oklawaha <2.3  f
## 10 oklawaha <2.3  m
## 11 oklawaha >2.3  f
## 12 oklawaha >2.3  m
## 13 trafford <2.3  f
## 14 trafford <2.3  m
## 15 trafford >2.3  f
## 16 trafford >2.3  m
```

$ %$ If you predict this in the model *without* `Gender`, you'll get the following:

```
pp = predict(gators.2, new, type = "p")
cbind(new, pp)
```

```
##        Lake Size Gender        Bird
## 1    george <2.3      f 0.029671502
## 2    george <2.3      m 0.029671502
## 3    george >2.3      f 0.081071082
## 4    george >2.3      m 0.081071082
## 5   hancock <2.3      f 0.070400215
## 6   hancock <2.3      m 0.070400215
## 7   hancock >2.3      f 0.140898571
## 8   hancock >2.3      m 0.140898571
## 9  oklawaha <2.3      f 0.008818267
## 10 oklawaha <2.3      m 0.008818267
## 11 oklawaha >2.3      f 0.029419560
## 12 oklawaha >2.3      m 0.029419560
## 13 trafford <2.3      f 0.035892547
## 14 trafford <2.3      m 0.035892547
## 15 trafford >2.3      f 0.108222209
## 16 trafford >2.3      m 0.108222209
##         Fish Invertebrate      Other
## 1  0.4521032   0.41285699 0.09380190
## 2  0.4521032   0.41285699 0.09380190
```

```
## 3  0.6574394   0.13967877 0.09791193
## 4  0.6574394   0.13967877 0.09791193
## 5  0.5353040   0.09309885 0.25374163
## 6  0.5353040   0.09309885 0.25374163
## 7  0.5701968   0.02307179 0.19400899
## 8  0.5701968   0.02307179 0.19400899
## 9  0.2581872   0.60189518 0.05387241
## 10 0.2581872   0.60189518 0.05387241
## 11 0.4584368   0.24864408 0.06866206
## 12 0.4584368   0.24864408 0.06866206
## 13 0.1842997   0.51683770 0.17420330
## 14 0.1842997   0.51683770 0.17420330
## 15 0.2957526   0.19296148 0.20066241
## 16 0.2957526   0.19296148 0.20066241
##       Reptile
## 1  0.01156641
## 2  0.01156641
## 3  0.02389880
## 4  0.02389880
## 5  0.04745531
## 6  0.04745531
## 7  0.07182382
## 8  0.07182382
## 9  0.07722691
## 10 0.07722691
## 11 0.19483754
## 12 0.19483754
## 13 0.08876673
## 14 0.08876673
## 15 0.20240133
## 16 0.20240133
```

Here, the predictions for each gender are *exactly the same*, because not having `Gender` in the model means that we take its effect to be *exactly zero*.

Alternatively, if you really thought the model with `Gender` was the better one, then you'd do this:

```
pp = predict(gators.1, new, type = "p")
cbind(new, pp)
```

```
##        Lake Size Gender        Bird
## 1    george <2.3      f 0.034528820
## 2    george <2.3      m 0.023983587
## 3    george >2.3      f 0.105463159
```

```
## 4    george >2.3      m 0.067888065
## 5   hancock <2.3      f 0.079170901
## 6   hancock <2.3      m 0.051120681
## 7   hancock >2.3      f 0.167234031
## 8   hancock >2.3      m 0.110233536
## 9  oklawaha <2.3      f 0.010861171
## 10 oklawaha <2.3      m 0.008370118
## 11 oklawaha >2.3      f 0.037755986
## 12 oklawaha >2.3      m 0.027647882
## 13 trafford <2.3      f 0.043846177
## 14 trafford <2.3      m 0.034440739
## 15 trafford >2.3      f 0.133999340
## 16 trafford >2.3      m 0.104507582
##         Fish Invertebrate      Other
## 1  0.3930846   0.46546988 0.09399531
## 2  0.5008716   0.37330933 0.09304268
## 3  0.5780952   0.17991064 0.10337131
## 4  0.6826531   0.13371938 0.09482794
## 5  0.5071007   0.10120258 0.26099808
## 6  0.6006664   0.07545143 0.24016625
## 7  0.5157600   0.02705184 0.19850489
## 8  0.6236556   0.02058879 0.18646783
## 9  0.2146058   0.63335376 0.05267686
## 10 0.3033923   0.56356778 0.05785201
## 11 0.3592072   0.27861325 0.06593316
## 12 0.4825354   0.23557146 0.06880557
## 13 0.1449092   0.54510349 0.16453404
## 14 0.2088069   0.49438416 0.18417900
## 15 0.2132364   0.21081238 0.18105124
## 16 0.3050805   0.18983929 0.20122886
##        Reptile
## 1  0.012921439
## 2  0.008792768
## 3  0.033159675
## 4  0.020911502
## 5  0.051527690
## 6  0.032595228
## 7  0.091449234
## 8  0.059054291
## 9  0.088502364
## 10 0.066817794
## 11 0.258490381
## 12 0.185439712
## 13 0.101607074
```

```
## 14 0.078189237
## 15 0.260900674
## 16 0.199343761
```

and this time there *is* an effect of gender, but it is smallish, as befits an effect that is not significant.

There were only 216 alligators total, which is a small sample size for this kind of thing, especially with all those parameters to estimate.

(g) What do you think is the most important way in which the lakes differ? (Hint: look at where the biggest predicted probabilities are.)

Solution

Here are the predictions again:

```
preds1
```

```
##       Lake Size        Bird      Fish
## 1   george <2.3 0.029671502 0.4521032
## 2   george >2.3 0.081071082 0.6574394
## 3  hancock <2.3 0.070400215 0.5353040
## 4  hancock >2.3 0.140898571 0.5701968
## 5 oklawaha <2.3 0.008818267 0.2581872
## 6 oklawaha >2.3 0.029419560 0.4584368
## 7 trafford <2.3 0.035892547 0.1842997
## 8 trafford >2.3 0.108222209 0.2957526
##   Invertebrate      Other    Reptile
## 1   0.41285699 0.09380190 0.01156641
## 2   0.13967877 0.09791193 0.02389880
## 3   0.09309885 0.25374163 0.04745531
## 4   0.02307179 0.19400899 0.07182382
## 5   0.60189518 0.05387241 0.07722691
## 6   0.24864408 0.06866206 0.19483754
## 7   0.51683770 0.17420330 0.08876673
## 8   0.19296148 0.20066241 0.20240133
```

Following my own hint: the preferred diet in George and Hancock lakes is fish, but the preferred diet in Oklawaha and Trafford lakes is (at least sometimes) invertebrates. That is to say, the preferred diet in those last two lakes is less likely to be invertebrates than it is in the first two (comparing for alligators of the same size). This is true for both large and small alligators, as it should be, since there is no interaction in the model. That will do, though you can also note that reptiles are more commonly found in the last two lakes, and birds sometimes appear in the diet in Hancock and Trafford but rarely in the other two lakes. Another way to think about this is to hold size constant and compare lakes (and then check that it applies to the

other size too). In this case, you'd find the biggest predictions among the first four rows, and then check that the pattern persists in the second four rows. (It does.) I think looking at predicted probabilities like this is the easiest way to see what the model is telling you. I also think that having a consistent recipe for doing predictions makes the process require a good deal less thought: get the values you want to predict for and store them in vectors with plural names; create a data frame for prediction using `crossing`, where the things inside are all "singular=plural"; run `predict` with model, new data and (if needed) type of value to predict, glue the predictions onto the new data. If you somehow mess up the creation of your new data frame (this typically happens by forgetting a variable that you should have included, or giving it the wrong name), `predict` will *silently* use the *original data* to predict from. Your only warning that this has happened is the size of the output; it should have as many rows as `new` did above, and the original data will typically have many more. (This happened to me just now. I recognized what the problem was, and how I would be able to fix it.)

(h) How would you describe the major difference between the diets of the small and large alligators?

Solution

Same idea: hold lake constant, and compare small and large, then check that your conclusion holds for the other lakes as it should. For example, in George Lake, the large alligators are more likely to eat fish, and less likely to eat invertebrates, compared to the small ones. The other food types are not that much different, though you might also note that birds appear more in the diets of large alligators than small ones. Does that hold in the other lakes? I think so, though there is less difference for fish in Hancock lake than the others (where invertebrates are rare for both sizes). Birds don't commonly appear in any alligator's diets, but where they do, they are commoner for large alligators than small ones.

17.6 How do you like your steak – the data

This question takes you through the data preparation for one of the other questions. You don't have to do this* question, but you may find it interesting or useful.

When you order a steak in a restaurant, the server will ask you how you would like it cooked, or to be precise, *how much* you would like it cooked: rare (hardly cooked at all), through medium rare, medium, medium well to well (which means "well done", so that the meat has only a little red to it). Could you guess how a person

likes their steak cooked, from some other information about them? The website link commissioned a survey where they asked a number of people how they preferred their steak, along with as many other things as they could think of to ask. (Many of the variables below are related to risk-taking, which was something the people designing the survey thought might have something to do with liking steak rare.) The variables of interest are all factors or true/false:

- `respondent_ID`: a ten-digit number identifying each person who responded to the survey.
- `lottery_a`: true if the respondent preferred lottery A with a small chance to win a lot of money, to lottery B, with a larger chance to win less money.
- `smoke`: true if the respondent is currently a smoker
- `alcohol`: true if the respondent at least occasionally drinks alcohol.
- `gamble`: true if the respondent likes to gamble (eg. betting on horse racing or playing the lottery)
- `skydiving`: true if the respondent has ever been skydiving.
- `speed`: true if the respondent likes to drive fast
- `cheated`: true if the respondent has ever cheated on a spouse or girlfriend/boyfriend
- `steak`: true if the respondent likes to eat steak
- `steak_prep` (response): how the respondent likes their steak cooked (factor, as described above, with 5 levels).
- `female`: true if the respondent is female
- `age`: age group, from 18–29 to 60+.
- `hhold_income`: household income group, from $0–24,999 to $150,000+.
- `educ`: highest level of education attained, from "less than high school" up to "graduate degree"
- `region`: region (of the US) that the respondent lives in (five values).

The data are in link.

(a) Read in the data and display the first few lines.

Solution
The usual:

```
my_url = "http://www.utsc.utoronto.ca/~butler/d29/steak.csv"
steak = read_csv(my_url)
```

```
## Parsed with column specification:
## cols(
##   respondent_id = col_double(),
##   lottery_a = col_logical(),
##   smoke = col_logical(),
##   alcohol = col_logical(),
##   gamble = col_logical(),
##   skydiving = col_logical(),
##   speed = col_logical(),
##   cheated = col_logical(),
##   steak = col_logical(),
##   steak_prep = col_character(),
##   female = col_logical(),
##   age = col_character(),
##   hhold_income = col_character(),
##   educ = col_character(),
##   region = col_character()
## )
```

```
steak
```

```
## # A tibble: 550 x 15
##    respondent_id lottery_a smoke alcohol
##            <dbl> <lgl>     <lgl> <lgl>
##  1    3237565956 FALSE     NA    NA
##  2    3234982343 TRUE      FALSE TRUE
##  3    3234973379 TRUE      FALSE TRUE
##  4    3234972383 FALSE     TRUE  TRUE
##  5    3234958833 FALSE     FALSE TRUE
##  6    3234955240 TRUE      FALSE FALSE
##  7    3234955097 TRUE      FALSE TRUE
##  8    3234955010 TRUE      FALSE TRUE
##  9    3234953052 TRUE      TRUE  TRUE
## 10    3234951249 FALSE     FALSE TRUE
## # ... with 540 more rows, and 11 more
## #   variables: gamble <lgl>,
## #   skydiving <lgl>, speed <lgl>,
## #   cheated <lgl>, steak <lgl>,
## #   steak_prep <chr>, female <lgl>,
## #   age <chr>, hhold_income <chr>,
## #   educ <chr>, region <chr>
```

(b) Run the function `complete.cases` on your data frame. This will

produce a lot of output. Pick three rows where `complete.cases` is TRUE and three where it is FALSE, and display those rows of the data frame. Looking at your results, what do you think `complete.cases` does? Explain briefly.

Solution

If you don't believe the whole output is long, well, here it is:

```
complete.cases(steak)
```

```
##   [1] FALSE  TRUE  TRUE  TRUE  TRUE  TRUE
##   [7] FALSE  TRUE  TRUE  TRUE  TRUE  TRUE
##  [13]  TRUE  TRUE  TRUE  TRUE  TRUE FALSE
##  [19]  TRUE FALSE  TRUE  TRUE FALSE  TRUE
##  [25]  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE
##  [31]  TRUE FALSE FALSE  TRUE FALSE FALSE
##  [37]  TRUE FALSE FALSE FALSE FALSE  TRUE
##  [43] FALSE  TRUE FALSE  TRUE  TRUE  TRUE
##  [49]  TRUE  TRUE FALSE FALSE  TRUE  TRUE
##  [55] FALSE  TRUE  TRUE  TRUE FALSE  TRUE
##  [61] FALSE  TRUE  TRUE FALSE  TRUE  TRUE
##  [67] FALSE FALSE FALSE  TRUE FALSE FALSE
##  [73] FALSE  TRUE  TRUE FALSE FALSE FALSE
##  [79]  TRUE FALSE FALSE  TRUE FALSE FALSE
##  [85] FALSE  TRUE  TRUE  TRUE FALSE FALSE
##  [91]  TRUE  TRUE  TRUE FALSE FALSE  TRUE
##  [97]  TRUE  TRUE FALSE  TRUE FALSE  TRUE
## [103] FALSE  TRUE  TRUE  TRUE FALSE  TRUE
## [109] FALSE  TRUE  TRUE FALSE FALSE  TRUE
## [115] FALSE  TRUE FALSE FALSE  TRUE  TRUE
## [121]  TRUE  TRUE  TRUE FALSE  TRUE  TRUE
## [127] FALSE FALSE FALSE  TRUE  TRUE  TRUE
## [133] FALSE FALSE  TRUE FALSE FALSE FALSE
## [139]  TRUE  TRUE  TRUE  TRUE FALSE  TRUE
## [145]  TRUE FALSE  TRUE  TRUE  TRUE FALSE
## [151]  TRUE  TRUE  TRUE  TRUE FALSE  TRUE
## [157]  TRUE  TRUE  TRUE FALSE FALSE  TRUE
## [163]  TRUE FALSE FALSE FALSE FALSE FALSE
## [169]  TRUE FALSE  TRUE FALSE  TRUE  TRUE
## [175] FALSE FALSE  TRUE  TRUE  TRUE FALSE
## [181] FALSE  TRUE FALSE FALSE  TRUE FALSE
## [187]  TRUE FALSE  TRUE  TRUE FALSE FALSE
## [193]  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE
## [199] FALSE  TRUE FALSE FALSE FALSE  TRUE
## [205]  TRUE FALSE  TRUE FALSE FALSE FALSE
```

```
## [211] FALSE FALSE  TRUE  TRUE  TRUE  TRUE
## [217] FALSE  TRUE  TRUE FALSE  TRUE  TRUE
## [223] FALSE  TRUE  TRUE FALSE  TRUE  TRUE
## [229]  TRUE  TRUE  TRUE FALSE  TRUE  TRUE
## [235]  TRUE  TRUE  TRUE  TRUE FALSE  TRUE
## [241]  TRUE  TRUE  TRUE  TRUE  TRUE FALSE
## [247] FALSE  TRUE FALSE  TRUE FALSE  TRUE
## [253] FALSE  TRUE FALSE  TRUE  TRUE  TRUE
## [259]  TRUE  TRUE  TRUE FALSE  TRUE  TRUE
## [265] FALSE  TRUE  TRUE  TRUE FALSE  TRUE
## [271]  TRUE  TRUE FALSE  TRUE FALSE FALSE
## [277]  TRUE FALSE  TRUE FALSE FALSE FALSE
## [283]  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE
## [289]  TRUE FALSE FALSE FALSE FALSE  TRUE
## [295]  TRUE  TRUE FALSE  TRUE FALSE  TRUE
## [301] FALSE  TRUE  TRUE  TRUE FALSE FALSE
## [307]  TRUE  TRUE FALSE  TRUE  TRUE  TRUE
## [313]  TRUE FALSE  TRUE  TRUE  TRUE  TRUE
## [319] FALSE  TRUE FALSE  TRUE FALSE  TRUE
## [325] FALSE FALSE  TRUE  TRUE FALSE  TRUE
## [331]  TRUE FALSE  TRUE  TRUE FALSE  TRUE
## [337]  TRUE  TRUE  TRUE  TRUE  TRUE FALSE
## [343] FALSE  TRUE FALSE FALSE  TRUE FALSE
## [349] FALSE  TRUE  TRUE  TRUE FALSE  TRUE
## [355]  TRUE FALSE FALSE  TRUE  TRUE  TRUE
## [361] FALSE  TRUE FALSE  TRUE  TRUE FALSE
## [367] FALSE  TRUE  TRUE  TRUE  TRUE FALSE
## [373] FALSE FALSE  TRUE FALSE FALSE  TRUE
## [379]  TRUE  TRUE  TRUE FALSE FALSE  TRUE
## [385]  TRUE  TRUE  TRUE  TRUE  TRUE FALSE
## [391] FALSE  TRUE FALSE FALSE  TRUE  TRUE
## [397]  TRUE  TRUE  TRUE  TRUE FALSE  TRUE
## [403]  TRUE FALSE  TRUE  TRUE  TRUE FALSE
## [409]  TRUE  TRUE  TRUE  TRUE FALSE  TRUE
## [415] FALSE  TRUE FALSE  TRUE FALSE FALSE
## [421]  TRUE FALSE FALSE  TRUE  TRUE  TRUE
## [427] FALSE  TRUE FALSE FALSE  TRUE  TRUE
## [433]  TRUE FALSE FALSE  TRUE  TRUE FALSE
## [439]  TRUE  TRUE FALSE FALSE  TRUE FALSE
## [445]  TRUE  TRUE  TRUE FALSE FALSE  TRUE
## [451]  TRUE FALSE  TRUE  TRUE FALSE  TRUE
## [457]  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE
## [463]  TRUE FALSE  TRUE  TRUE  TRUE FALSE
## [469]  TRUE  TRUE FALSE  TRUE FALSE FALSE
```

```
## [475]  TRUE  TRUE FALSE FALSE  TRUE  TRUE
## [481] FALSE FALSE  TRUE  TRUE  TRUE  TRUE
## [487]  TRUE FALSE  TRUE FALSE  TRUE FALSE
## [493] FALSE  TRUE  TRUE  TRUE  TRUE  TRUE
## [499] FALSE  TRUE FALSE FALSE  TRUE FALSE
## [505]  TRUE FALSE FALSE  TRUE FALSE  TRUE
## [511] FALSE  TRUE FALSE  TRUE  TRUE FALSE
## [517]  TRUE  TRUE FALSE FALSE  TRUE  TRUE
## [523] FALSE FALSE  TRUE  TRUE  TRUE FALSE
## [529] FALSE FALSE  TRUE FALSE FALSE  TRUE
## [535]  TRUE FALSE FALSE  TRUE  TRUE FALSE
## [541] FALSE FALSE  TRUE  TRUE  TRUE  TRUE
## [547] FALSE  TRUE FALSE FALSE
```

It doesn't matter which three TRUE and which three FALSE values you pick (as long as they *are* TRUE or FALSE as you claim).

I'll pick rows 496, 497, and 498 for the TRUE rows, and 540, 541 and 542 for the FALSE ones. Let's assemble these rows into a vector and use `slice` to display the rows with these numbers:

```
rows = c(496, 497, 498, 540, 541, 542)
rows
```

```
## [1] 496 497 498 540 541 542
```

Like this:

```
steak %>% slice(rows)
```

```
## # A tibble: 6 x 15
##   respondent_id lottery_a smoke alcohol
##           <dbl> <lgl>     <lgl> <lgl>
## 1    3234776895 FALSE     FALSE FALSE
## 2    3234776815 TRUE      TRUE  TRUE
## 3    3234776702 FALSE     FALSE FALSE
## 4    3234763650 TRUE      FALSE FALSE
## 5    3234763171 TRUE      FALSE TRUE
## 6    3234762715 FALSE     FALSE FALSE
## # ... with 11 more variables: gamble <lgl>,
## #   skydiving <lgl>, speed <lgl>,
## #   cheated <lgl>, steak <lgl>,
## #   steak_prep <chr>, female <lgl>,
## #   age <chr>, hhold_income <chr>,
## #   educ <chr>, region <chr>
```

(c) What's the difference between the rows where `complete.cases` is TRUE and the rows where it is false?

Solution

The rows where `complete.cases` is FALSE have one (or more) missing values in them; where `complete.cases` is TRUE the rows have no missing values. (Depending on the rows you choose, you may not see the missing value(s), as I didn't.) Extra: this is a bit tricky to investigate more thoroughly, because the text variables might have missing values in them, and they won't show up unless we turn them into a factor first:

```
steak %>% mutate_if(is.character, factor) %>%
    summary()
```

```
##  respondent_id       lottery_a
##  Min.   :3.235e+09   Mode :logical
##  1st Qu.:3.235e+09   FALSE:279
##  Median :3.235e+09   TRUE :267
##  Mean   :3.235e+09   NA's :4
##  3rd Qu.:3.235e+09
##  Max.   :3.238e+09
##
##    smoke          alcohol
##  Mode :logical   Mode :logical
##  FALSE:453       FALSE:125
##  TRUE :84        TRUE :416
##  NA's :13        NA's :9
##
##
##
##    gamble        skydiving
##  Mode :logical   Mode :logical
##  FALSE:280       FALSE:502
##  TRUE :257       TRUE :36
##  NA's :13        NA's :12
##
##
##
##    speed          cheated
##  Mode :logical   Mode :logical
##  FALSE:59        FALSE:447
##  TRUE :480       TRUE :92
##  NA's :11        NA's :11
##
##
##
##    steak               steak_prep
```

```
##  Mode :logical   Medium     :132
##  FALSE:109       Medium rare:166
##  TRUE :430       Medium Well: 75
##  NA's :11        Rare       : 23
##                  Well       : 36
##                  NA's       :118
##
##    female           age
##  Mode :logical   18-29:110
##  FALSE:246       30-44:133
##  TRUE :268       45-60:140
##  NA's :36        >60  :131
##                  NA's : 36
##
##
##               hhold_income
##  $0 - $24,999       : 51
##  $100,000 - $149,999: 76
##  $150,000+          : 54
##  $25,000 - $49,999  : 77
##  $50,000 - $99,999  :172
##  NA's               :120
##
##                                 educ
##  Bachelor degree                  :174
##  Graduate degree                  :133
##  High school degree               : 39
##  Less than high school degree     :  2
##  Some college or Associate degree:164
##  NA's                             : 38
##
##                 region
##  Pacific           : 91
##  South Atlantic    : 88
##  East North Central: 86
##  Middle Atlantic   : 72
##  West North Central: 42
##  (Other)           :133
##  NA's              : 38
```

There are missing values everywhere. What the `_if` functions do is to do something for each column where the first thing is true: here, if the column is text, then replace it by the factor version of itself. This makes for a better summary, one that shows how many observations

are in each category, and, more important for us, how many are missing (a lot).

All right, so there are 15 columns, so let's investigate missingness in our rows by looking at the columns 1 through 8 and then 9 through 15, so they all fit on the screen. Recall that you can `select` columns by number:

```
steak %>% select(1:8) %>% slice(rows)
```

```
## # A tibble: 6 x 8
##   respondent_id lottery_a smoke alcohol
##           <dbl> <lgl>     <lgl> <lgl>
## 1    3234776895 FALSE     FALSE FALSE
## 2    3234776815 TRUE      TRUE  TRUE
## 3    3234776702 FALSE     FALSE FALSE
## 4    3234763650 TRUE      FALSE FALSE
## 5    3234763171 TRUE      FALSE TRUE
## 6    3234762715 FALSE     FALSE FALSE
## # ... with 4 more variables: gamble <lgl>,
## #   skydiving <lgl>, speed <lgl>,
## #   cheated <lgl>
```

and

```
steak %>% select(9:15) %>% slice(rows)
```

```
## # A tibble: 6 x 7
##   steak steak_prep female age   hhold_income
##   <lgl> <chr>      <lgl>  <chr> <chr>
## 1 TRUE  Medium ra~ TRUE   45-60 $0 - $24,999
## 2 TRUE  Medium     TRUE   45-60 $150,000+
## 3 TRUE  Medium ra~ TRUE   >60   $50,000 - $~
## 4 FALSE <NA>       FALSE  45-60 $100,000 - ~
## 5 TRUE  Medium     FALSE  >60   <NA>
## 6 FALSE <NA>       FALSE  18-29 $50,000 - $~
## # ... with 2 more variables: educ <chr>,
## #   region <chr>
```

In this case, the first three rows have no missing values anywhere, and the last three rows have exactly one missing value. This corresponds to what we would expect, with `complete.cases` identifying rows that have any missing values.

(d) Obtain a new data frame that contains only the rows without missing values (from the data frame you read in from the file). Hints: save the result of `complete.cases` in a variable first; `filter` can take anything that produces a true or a false for each row, and will return the rows for which the thing it was fed was true.

Solution

The hint, together with the previous part, suggests that the thing fed into `filter` should be based on `complete.cases(steak)`. Create a variable containing the results of `complete.cases` first, then use it in the `filter`:

```
cc = complete.cases(steak)
steak.complete = steak %>% filter(cc)
```

A quick check that we got rid of the missing values:

```
steak.complete
```

```
## # A tibble: 331 x 15
##    respondent_id lottery_a smoke alcohol
##            <dbl> <lgl>     <lgl> <lgl>
##  1    3234982343 TRUE      FALSE TRUE
##  2    3234973379 TRUE      FALSE TRUE
##  3    3234972383 FALSE     TRUE  TRUE
##  4    3234958833 FALSE     FALSE TRUE
##  5    3234955240 TRUE      FALSE FALSE
##  6    3234955010 TRUE      FALSE TRUE
##  7    3234953052 TRUE      TRUE  TRUE
##  8    3234951249 FALSE     FALSE TRUE
##  9    3234948883 FALSE     FALSE TRUE
## 10    3234948197 TRUE      FALSE FALSE
## # ... with 321 more rows, and 11 more
## #   variables: gamble <lgl>,
## #   skydiving <lgl>, speed <lgl>,
## #   cheated <lgl>, steak <lgl>,
## #   steak_prep <chr>, female <lgl>,
## #   age <chr>, hhold_income <chr>,
## #   educ <chr>, region <chr>
```

There are no missing values *there*. Of course, this is not a proof, and there might be some missing values further down, but at least it suggests that we might be good.

For proof, this is the easiest way I know:

```
steak.complete %>% mutate_if(is.character, factor) %>%
    summary()
```

```
##  respondent_id       lottery_a
##  Min.   :3.235e+09   Mode :logical
##  1st Qu.:3.235e+09   FALSE:171
##  Median :3.235e+09   TRUE :160
```

```
##  Mean   :3.235e+09
##  3rd Qu.:3.235e+09
##  Max.   :3.235e+09
##
##    smoke          alcohol
##  Mode :logical   Mode :logical
##  FALSE:274       FALSE:65
##  TRUE :57        TRUE :266
##
##
##
##
##    gamble        skydiving
##  Mode :logical   Mode :logical
##  FALSE:158       FALSE:308
##  TRUE :173       TRUE :23
##
##
##
##
##    speed          cheated
##  Mode :logical   Mode :logical
##  FALSE:28        FALSE:274
##  TRUE :303       TRUE :57
##
##
##
##
##   steak               steak_prep
##  Mode:logical   Medium     :109
##  TRUE:331       Medium rare:128
##                 Medium Well: 56
##                 Rare       : 18
##                 Well       : 20
##
##
##    female           age
##  Mode :logical   18-29:70
##  FALSE:174       30-44:93
##  TRUE :157       45-60:86
##                  >60  :82
##
##
##
```

```
##                hhold_income
##  $0 - $24,999       : 37
##  $100,000 - $149,999: 66
##  $150,000+          : 39
##  $25,000 - $49,999  : 55
##  $50,000 - $99,999  :134
##
##
##                                 educ
##  Bachelor degree                  :120
##  Graduate degree                  : 86
##  High school degree               : 20
##  Less than high school degree     :  1
##  Some college or Associate degree:104
##
##
##                  region
##  South Atlantic    :68
##  Pacific           :57
##  East North Central:48
##  Middle Atlantic   :46
##  West North Central:29
##  Mountain          :24
##  (Other)           :59
```

If there were any missing values, they would be listed on the end of the counts of observations for each level, or on the bottom of the five-number sumamries. But there aren't. So here's your proof.

(e) Write the data into a `.csv` file, with a name like `steak1.csv`. Open this file in a spreadsheet and (quickly) verify that you have the right columns and no missing values.

Solution
This is `write_csv`:

```
write_csv(steak.complete, "steak1.csv")
```

Open up Excel, or whatever you have, and take a look. You should have all the right columns, and, scrolling down, no visible missing values.

17.7 *Crimes in San Francisco – the data*

The data in link is a huge dataset of crimes committed in San Francisco between 2003 and 2015. The variables are:

- `Dates`: the date and time of the crime
- `Category`: the category of crime, eg. "larceny" or "vandalism" (response).
- `Descript`: detailed description of crime.
- `DayOfWeek`: the day of the week of the crime.
- `PdDistrict`: the name of the San Francisco Police Department district in which the crime took place.
- `Resolution`: how the crime was resolved
- `Address`: approximate street address of crime
- `X`: longitude
- `Y`: latitude

Our aim is to see whether the category of crime depends on the day of the week and the district in which it occurred. However, there are a lot of crime categories, so we will focus on the top four "interesting" ones, which we will have to discover.

(a) Read in the data and verify that you have these columns and a lot of rows. (The data may take a moment to read in. You will see why.)

Solution

```
my_url = "http://www.utsc.utoronto.ca/~butler/d29/sfcrime.csv"
sfcrime = read_csv(my_url)
```

```
## Parsed with column specification:
## cols(
##   Dates = col_datetime(format = ""),
##   Category = col_character(),
##   Descript = col_character(),
##   DayOfWeek = col_character(),
##   PdDistrict = col_character(),
##   Resolution = col_character(),
##   Address = col_character(),
##   X = col_double(),
##   Y = col_double()
## )
```

```
sfcrime
```

```
## # A tibble: 878,049 x 9
##    Dates               Category Descript
##    <dttm>              <chr>    <chr>
##  1 2015-05-13 23:53:00 WARRANTS WARRANT~
##  2 2015-05-13 23:53:00 OTHER O~ TRAFFIC~
##  3 2015-05-13 23:33:00 OTHER O~ TRAFFIC~
##  4 2015-05-13 23:30:00 LARCENY~ GRAND T~
##  5 2015-05-13 23:30:00 LARCENY~ GRAND T~
##  6 2015-05-13 23:30:00 LARCENY~ GRAND T~
##  7 2015-05-13 23:30:00 VEHICLE~ STOLEN ~
##  8 2015-05-13 23:30:00 VEHICLE~ STOLEN ~
##  9 2015-05-13 23:00:00 LARCENY~ GRAND T~
## 10 2015-05-13 23:00:00 LARCENY~ GRAND T~
## # ... with 878,039 more rows, and 6 more
## #   variables: DayOfWeek <chr>,
## #   PdDistrict <chr>, Resolution <chr>,
## #   Address <chr>, X <dbl>, Y <dbl>
```

Those columns indeed, and pushing a million rows! That's why it took so long!

There are also 39 categories of crime, so we need to cut that down some. There are only ten districts, however, so we should be able to use that as is.

(b) How is the response variable here different to the one in the question about steak preferences (and therefore why would `multinom` from package `nnet` be the method of choice)?

Solution

Steak preferences have a natural order, while crime categories do not. Since they are unordered, `multinom` is better than `polr`.

(c) Find out which crime categories there are, and arrange them in order of how many crimes there were in each category.

Solution

This can be the one you know, a group-by and summarize, followed by `arrange` to sort:

```
sfcrime %>% group_by(Category) %>% summarize(count = n()) %>%
    arrange(desc(count))
```

```
## # A tibble: 39 x 2
##    Category        count
##    <chr>           <int>
##  1 LARCENY/THEFT  174900
```

```
##  2 OTHER OFFENSES 126182
##  3 NON-CRIMINAL    92304
##  4 ASSAULT         76876
##  5 DRUG/NARCOTIC   53971
##  6 VEHICLE THEFT   53781
##  7 VANDALISM       44725
##  8 WARRANTS        42214
##  9 BURGLARY        36755
## 10 SUSPICIOUS OCC  31414
## # ... with 29 more rows
```

or this one does the same thing and saves a step:

```
sfcrime %>% count(Category) %>% arrange(desc(n))
```

```
## # A tibble: 39 x 2
##    Category            n
##    <chr>           <int>
##  1 LARCENY/THEFT  174900
##  2 OTHER OFFENSES 126182
##  3 NON-CRIMINAL    92304
##  4 ASSAULT         76876
##  5 DRUG/NARCOTIC   53971
##  6 VEHICLE THEFT   53781
##  7 VANDALISM       44725
##  8 WARRANTS        42214
##  9 BURGLARY        36755
## 10 SUSPICIOUS OCC  31414
## # ... with 29 more rows
```

For this one, do the `count` step first, to see what you get. It produces a two-column data frame with the column of counts called `n`. So now you know that the second line has to be `arrange(desc(n))`, whereas before you tried `count`, all you knew is that it was arrange-desc-something.

You need to sort in descending order so that the categories you want to see actually do appear at the top.

(d) Which are the four most frequent "interesting" crime categories, that is to say, not including "other offenses" and "non-criminal"? Get them into a vector called `my.crimes`. See if you can find a way of doing this that doesn't involve typing them in (for full marks).

Solution

The most frequent interesting ones are, in order, larceny-theft, assault, drug-narcotic and vehicle theft. The fact that "other offenses" is

so big indicates that there are a lot of possible crimes out there, and even 39 categories of crime isn't enough. "Non-criminal" means, I think, that the police were called, but on arriving at the scene, they found that no law had been broken. I think the easy way to get the "top four" crimes out is to pull them out of the data frame that `count` produces. They are rows 1, 4, 5 and 6, so add a `slice` to your pipeline:

```
my.rows = c(1, 4, 5, 6)
my.crimes = sfcrime %>% count(Category) %>% arrange(desc(n)) %>%
    slice(my.rows) %>% pull(Category)
my.crimes
```

```
## [1] "LARCENY/THEFT" "ASSAULT"
## [3] "DRUG/NARCOTIC" "VEHICLE THEFT"
```

I just want the `Category` column (as a vector), and `pull` is the way to get that. (If I don't do `pull`, I get a data frame.)

If you can't think of anything, just type them into a vector with `c`, or better, copy-paste them from the console. But that's a last resort, and would cost you a point. If you do this, they have to match exactly, UPPERCASE and all.

(e) (Digression, but needed for the next part.) The R vector `letters` contains the lowercase letters from `a` to `z`. Consider the vector `('a','m',3,'Q')`. Some of these are found amongst the lowercase letters, and some not. Type these into a vector `v` and explain briefly why `v %in% letters` produces what it does.

Solution

This is the ultimate "try it and see":

```
v = c("a", "m", 3, "Q")
v %in% letters
```

```
## [1]  TRUE  TRUE FALSE FALSE
```

The first two elements of the answer are `TRUE` because lowercase-a and lowercase-m can be found in the lowercase letters somewhere. The other two are false because the number 3 and the uppercase-Q cannot be found anywhere in the lowercase letters.

The name is `%in%` because it's asking whether each element of the first vector (one at a time) is *in* the set defined by the second thing: "is `a` a lowercase letter?" ... is "`Q` a lowercase letter?" and getting the answers "yes, yes, no, no".

(f) We are going to `filter` only the rows of our data frame that have one of the crimes in `my.crimes` as their `Category`. Also, `select`

only the columns `Category`, `DayOfWeek` and `PdDistrict`. Save the resulting data frame and display its structure. (You should have a lot fewer rows than you did before.)

Solution

The hard part about this is to get the inputs to `%in%` the right way around. We are testing the things in `Category` one at a time for membership in the set in `my.crimes`, so this:

```
sfcrimea = sfcrime %>% filter(Category %in% my.crimes) %>%
    select(c(Category, DayOfWeek, PdDistrict))
sfcrimea
```

```
## # A tibble: 359,528 x 3
##    Category      DayOfWeek PdDistrict
##    <chr>         <chr>     <chr>
##  1 LARCENY/THEFT Wednesday NORTHERN
##  2 LARCENY/THEFT Wednesday PARK
##  3 LARCENY/THEFT Wednesday INGLESIDE
##  4 VEHICLE THEFT Wednesday INGLESIDE
##  5 VEHICLE THEFT Wednesday BAYVIEW
##  6 LARCENY/THEFT Wednesday RICHMOND
##  7 LARCENY/THEFT Wednesday CENTRAL
##  8 LARCENY/THEFT Wednesday CENTRAL
##  9 LARCENY/THEFT Wednesday NORTHERN
## 10 ASSAULT       Wednesday INGLESIDE
## # ... with 359,518 more rows
```

That didn't work. Remember what this was? This is because `MASS` is still loaded:

```
search()
```

```
##  [1] ".GlobalEnv"        "package:bindrcpp"
##  [3] "package:forcats"   "package:stringr"
##  [5] "package:dplyr"     "package:purrr"
##  [7] "package:readr"     "package:tidyr"
##  [9] "package:tibble"    "package:ggplot2"
## [11] "package:tidyverse" "package:nnet"
## [13] "package:stats"     "package:graphics"
## [15] "package:grDevices" "package:utils"
## [17] "package:datasets"  "package:methods"
## [19] "Autoloads"         "package:base"
```

so we need to get rid of `MASS`:

```
detach("package:MASS", unload = T)
```

```
## Error in detach("package:MASS", unload = T): invalid 'name' argument
```

and try again:

```
sfcrimea = sfcrime %>% filter(Category %in% my.crimes) %>%
    select(c(Category, DayOfWeek, PdDistrict))
sfcrimea
```

```
## # A tibble: 359,528 x 3
##    Category      DayOfWeek PdDistrict
##    <chr>         <chr>     <chr>
##  1 LARCENY/THEFT Wednesday NORTHERN
##  2 LARCENY/THEFT Wednesday PARK
##  3 LARCENY/THEFT Wednesday INGLESIDE
##  4 VEHICLE THEFT Wednesday INGLESIDE
##  5 VEHICLE THEFT Wednesday BAYVIEW
##  6 LARCENY/THEFT Wednesday RICHMOND
##  7 LARCENY/THEFT Wednesday CENTRAL
##  8 LARCENY/THEFT Wednesday CENTRAL
##  9 LARCENY/THEFT Wednesday NORTHERN
## 10 ASSAULT       Wednesday INGLESIDE
## # ... with 359,518 more rows
```

I had trouble thinking of a good name for this one, so I put an "a" on the end. (I would have used a number, but I prefer those for models.)

Down to a "mere" 359,000 rows. Don't be stressed that the `Category` factor still has 39 levels (the original 39 crime categories); only four of them have any data in them:

```
sfcrimea %>% count(Category)
```

```
## # A tibble: 4 x 2
##   Category           n
##   <chr>          <int>
## 1 ASSAULT        76876
## 2 DRUG/NARCOTIC  53971
## 3 LARCENY/THEFT 174900
## 4 VEHICLE THEFT  53781
```

So all of the crimes that are left are one of the four Categories we want to look at.

(g) Save these data in a file `sfcrime1.csv`.

Solution

This is `write_csv` again:

```
write_csv(sfcrimea, "sfcrime1.csv")
```

17.8 How do you like your steak?

When you order a steak in a restaurant, the server will ask you how you would like it cooked, or to be precise, *how much* you would like it cooked: rare (hardly cooked at all), through medium rare, medium, medium well to well (which means "well done", so that the meat has only a little red to it). Could you guess how a person likes their steak cooked, from some other information about them? The website link commissioned a survey where they asked a number of people how they preferred their steak, along with as many other things as they could think of to ask. (Many of the variables below are related to risk-taking, which was something the people designing the survey thought might have something to do with liking steak rare.) The variables of interest are all factors or true/false:

- `respondent_ID`: a ten-digit number identifying each person who responded to the survey.
- `lottery_a`: true if the respondent preferred lottery A with a small chance to win a lot of money, to lottery B, with a larger chance to win less money.
- `smoke`: true if the respondent is currently a smoker
- `alcohol`: true if the respondent at least occasionally drinks alcohol.
- `gamble`: true if the respondent likes to gamble (eg. betting on horse racing or playing the lottery)
- `skydiving`: true if the respondent has ever been skydiving.
- `speed`: true if the respondent likes to drive fast
- `cheated`: true if the respondent has ever cheated on a spouse or girlfriend/boyfriend
- `steak`: true if the respondent likes to eat steak
- `steak_prep` (response): how the respondent likes their steak cooked (factor, as described above, with 5 levels).
- `female`: true if the respondent is female
- `age`: age group, from 18–29 to 60+.
- `hhold_income`: household income group, from $0–24,999 to $150,000+.
- `educ`: highest level of education attained, from "less than high school" up to "graduate degree"

- `region`: region (of the US) that the respondent lives in (five values).

The data are in link. This is the cleaned-up data from a previous question, with the missing values removed.

(a) Read in the data and display the first few lines.

Solution
The usual:

```
steak = read_csv("steak1.csv")
```

```
## Parsed with column specification:
## cols(
##   respondent_id = col_double(),
##   lottery_a = col_logical(),
##   smoke = col_logical(),
##   alcohol = col_logical(),
##   gamble = col_logical(),
##   skydiving = col_logical(),
##   speed = col_logical(),
##   cheated = col_logical(),
##   steak = col_logical(),
##   steak_prep = col_character(),
##   female = col_logical(),
##   age = col_character(),
##   hhold_income = col_character(),
##   educ = col_character(),
##   region = col_character()
## )
```

```
head(steak)
```

```
## # A tibble: 6 x 15
##   respondent_id lottery_a smoke alcohol
##           <dbl> <lgl>     <lgl> <lgl>
## 1    3234982343 TRUE      FALSE TRUE
## 2    3234973379 TRUE      FALSE TRUE
## 3    3234972383 FALSE     TRUE  TRUE
## 4    3234958833 FALSE     FALSE TRUE
## 5    3234955240 TRUE      FALSE FALSE
## 6    3234955010 TRUE      FALSE TRUE
## # ... with 11 more variables: gamble <lgl>,
## #   skydiving <lgl>, speed <lgl>,
## #   cheated <lgl>, steak <lgl>,
```

```
## #   steak_prep <chr>, female <lgl>,
## #   age <chr>, hhold_income <chr>,
## #   educ <chr>, region <chr>
```

We should also (re-)load `MASS`, since we'll be needing it:

```
library(MASS)
```

```
##
## Attaching package: 'MASS'

## The following object is masked from 'package:dplyr':
##
##     select
```

(b) We are going to predict `steak_prep` from some of the other variables. Why is the model-fitting function `polr` from package `MASS` the best choice for these data (alternatives being `glm` and `multinom` from package `nnet`)?

Solution

It all depends on the kind of response variable. We have a response variable with five ordered levels from Rare to Well. There are more than two levels (it is more than a "success" and "failure"), which rules out `glm`, and the levels are ordered, which rules out `multinom`. As we know, `polr` handles an ordered response, so it is the right choice.

(c) What are the levels of `steak_prep`, *in the order that R thinks they are in?* If they are not in a sensible order, create an ordered factor where the levels are in a sensible order.

Solution

This is the most direct way to find out:

```
preps = steak %>% distinct(steak_prep) %>% pull(steak_prep)
preps
```

```
## [1] "Medium rare" "Rare"        "Medium"
## [4] "Medium Well" "Well"
```

$ %$ %$ %$

This is almost the right order (`distinct` uses the order in the data frame). We just need to switch the first two around, and then we'll be done:

```
preps1 = preps[c(2, 1, 3, 4, 5)]
preps1
```

```
## [1] "Rare"        "Medium rare" "Medium"
## [4] "Medium Well" "Well"
```

If you used `count`, there's a bit more work to do:

```
preps2 = steak %>% count(steak_prep) %>% pull(steak_prep)
preps2
```

```
## [1] "Medium"      "Medium rare" "Medium Well"
## [4] "Rare"        "Well"
```

because `count` puts them in alphabetical order, so:

```
preps3 = preps2[c(4, 2, 1, 3, 5)]
preps3
```

```
## [1] "Rare"        "Medium rare" "Medium"
## [4] "Medium Well" "Well"
```

These use the idea in the attitudes-to-abortion question: create a vector of the levels in the *right* order, then create an ordered factor with `ordered()`. If you like, you can type the levels in the right order (I won't penalize you for that here), but it's really better to get the levels without typing or copying-pasting, so that you don't make any silly errors copying them (which will mess everything up later).

So now I create my ordered response:

```
steak = steak %>% mutate(steak_prep_ord = ordered(steak_prep,
    preps1))
```

or using one of the other `preps` vectors containing the levels in the correct order. $ %$ %$

As far as `polr` is concerned, it doesn't matter whether I start at `Rare` and go "up", or start at `Well` and go "down". So if you do it the other way around, that's fine. As long as you get the levels in a sensible order, you're good.

(d) Fit a model predicting preferred steak preparation in an ordinal logistic regression from `educ`, `female` and `lottery_a`. This ought to be easy from your previous work, but you have to be careful about one thing. No need to print out the results.

Solution

The thing you have to be careful about is that you use the *ordered* factor that you just created as the response:

```
steak.1 = polr(steak_prep_ord ~ educ + female +
    lottery_a, data = steak)
```

(e) Run `drop1` on your fitted model, with `test="Chisq"`. Which explanatory variable should be removed first, if any? Bear in mind that the variable with the *smallest* AIC should come out first, in case your table doesn't get printed in order.

Solution

This:

```
drop1(steak.1, test = "Chisq")
```

```
## Single term deletions
##
## Model:
## steak_prep_ord ~ educ + female + lottery_a
##           Df    AIC    LRT Pr(>Chi)
## <none>       910.69
## educ       4 912.10 9.4107  0.05162 .
## female     1 908.70 0.0108  0.91715
## lottery_a  1 909.93 1.2425  0.26498
## ---
## Signif. codes:
##   0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1  ' ' 1
```

My table is indeed out of order (which is why I warned you about it, in case that happens to you as well). The smallest AIC goes with `female`, which also has a very non-significant P-value, so this one should come out first.

(f) Remove the variable that should come out first, using `update`. (If all the variables should stay, you can skip this part.)

Solution

You could type or copy-paste the whole model again, but `update` is quicker:

```
steak.2 = update(steak.1, . ~ . - female)
```

That's all.

I wanted to get some support for my `drop1` above (since I was a bit worried about those out-of-order rows). Now that we have fitted a model with `female` and one without, we can compare them using `anova`:

```
anova(steak.2, steak.1, test = "Chisq")
```

```
## Likelihood ratio tests of ordinal regression models
##
```

```
## Response: steak_prep_ord
##                        Model Resid. df
## 1          educ + lottery_a       322
## 2 educ + female + lottery_a       321
##   Resid. Dev   Test    Df  LR stat.
## 1   890.7028
## 2   890.6920 1 vs 2     1 0.0108221
##     Pr(Chi)
## 1
## 2 0.9171461
```

Don't get taken in by that "LR stat" on the end of the first row of the output table; the P-value wrapped onto the second line, and is in fact exactly the same as in the `drop1` output (it is doing exactly the same test). As non-significant as you could wish for.

I was curious about whether either of the other x's could come out now:

```r
drop1(steak.2, test = "Chisq")
```

```
## Single term deletions
##
## Model:
## steak_prep_ord ~ educ + lottery_a
##           Df    AIC    LRT Pr(>Chi)
## <none>       908.70
## educ       4 910.13 9.4299  0.05121 .
## lottery_a  1 907.96 1.2599  0.26167
## ---
## Signif. codes:
##   0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1  ' ' 1
```

`lottery_a` should come out, but `educ` is edging towards significance.

(g) Using the best model that you have so far, predict the probabilities of preferring each different steak preparation (method of cooking) for each combination of the variables that remain. (Some of the variables are TRUE and FALSE rather than factors. Bear this in mind.) Describe the effects of each variable on the predicted probabilities, if any. Note that there is exactly one person in the study whose educational level is "less than high school".

Solution

Again, I'm leaving it to you to follow all the steps. My variables remaining are `educ` and `lottery_a`. The second of these is a logical, TRUE and FALSE, so I just filled in the two values:

```
educs = steak %>% distinct(educ) %>% pull(educ)
lottery_as = c(FALSE, TRUE)

steak.new = crossing(educ = educs, lottery_a = lottery_as)
steak.new
```

```
## # A tibble: 10 x 2
##    educ                              lottery_a
##    <chr>                             <lgl>
##  1 Bachelor degree                   FALSE
##  2 Bachelor degree                   TRUE
##  3 Graduate degree                   FALSE
##  4 Graduate degree                   TRUE
##  5 High school degree                FALSE
##  6 High school degree                TRUE
##  7 Less than high school degree      FALSE
##  8 Less than high school degree      TRUE
##  9 Some college or Associate degree FALSE
## 10 Some college or Associate degree TRUE
```

My best model so far is the one I called `steak.2`, without `female`, so this. As before, if you forget what `type` should be, put in *something*, and it will tell you what your choices are:

```
p = predict(steak.2, steak.new, type = "probs")
cbind(steak.new, p)
```

```
##                                 educ
## 1                   Bachelor degree
## 2                   Bachelor degree
## 3                   Graduate degree
## 4                   Graduate degree
## 5                High school degree
## 6                High school degree
## 7      Less than high school degree
## 8      Less than high school degree
## 9  Some college or Associate degree
## 10 Some college or Associate degree
##    lottery_a         Rare  Medium rare
## 1      FALSE 5.191314e-02 3.832425e-01
## 2       TRUE 4.181795e-02 3.386213e-01
## 3      FALSE 7.954891e-02 4.691799e-01
## 4       TRUE 6.444488e-02 4.277303e-01
## 5      FALSE 5.280801e-02 3.867857e-01
## 6       TRUE 4.254661e-02 3.421527e-01
## 7      FALSE 8.507263e-08 1.111878e-06
```

```
## 8       TRUE 6.780714e-08 8.862222e-07
## 9      FALSE 5.499694e-02 3.951981e-01
## 10      TRUE 4.433010e-02 3.505796e-01
##          Medium  Medium Well       Well
## 1  3.347391e-01 1.719854e-01 0.05811984
## 2  3.468434e-01 2.008621e-01 0.07185531
## 3  2.920587e-01 1.215881e-01 0.03762435
## 4  3.158550e-01 1.452132e-01 0.04675658
## 5  3.334805e-01 1.697854e-01 0.05714038
## 6  3.461457e-01 1.984932e-01 0.07066175
## 7  4.001373e-06 1.997966e-05 0.99997482
## 8  3.189298e-06 1.592489e-05 0.99997993
## 9  3.303218e-01 1.646121e-01 0.05487110
## 10 3.442957e-01 1.929005e-01 0.06789415
```

I find this hard to read, so I'm going to round off those predictions. Three or four decimals seems to be sensible:

```r
cbind(steak.new, round(p, 3))
```

```
##                                 educ
## 1                   Bachelor degree
## 2                   Bachelor degree
## 3                   Graduate degree
## 4                   Graduate degree
## 5                High school degree
## 6                High school degree
## 7      Less than high school degree
## 8      Less than high school degree
## 9  Some college or Associate degree
## 10 Some college or Associate degree
##    lottery_a  Rare Medium rare Medium
## 1      FALSE 0.052       0.383  0.335
## 2       TRUE 0.042       0.339  0.347
## 3      FALSE 0.080       0.469  0.292
## 4       TRUE 0.064       0.428  0.316
## 5      FALSE 0.053       0.387  0.333
## 6       TRUE 0.043       0.342  0.346
## 7      FALSE 0.000       0.000  0.000
## 8       TRUE 0.000       0.000  0.000
## 9      FALSE 0.055       0.395  0.330
## 10      TRUE 0.044       0.351  0.344
##    Medium Well  Well
## 1        0.172 0.058
## 2        0.201 0.072
```

```
## 3        0.122 0.038
## 4        0.145 0.047
## 5        0.170 0.057
## 6        0.198 0.071
## 7        0.000 1.000
## 8        0.000 1.000
## 9        0.165 0.055
## 10       0.193 0.068
```

Say something about the effect of changing educational level on the predictions, and say something about the effect of favouring Lottery A vs. not. I don't much mind what: you can say that there is not much effect (of either variable), or you can say something like "people with a graduate degree are slightly more likely to like their steak rare and less likely to like it well done" (for education level) and "people who preferred Lottery A are slightly less likely to like their steak rare and slightly more likely to like it well done" (for effect of Lottery A). You can see these by comparing the first five rows to assess the effect of education (or the last five rows, if you prefer), and you can compare eg. rows 1 and 6 to assess the effect of Lottery A (compare two lines with the *same* educational level but *different* preferences re Lottery A).

I would keep away from saying anything about education level "less than high school", since this entire level is represented by exactly one person.

(h) Is it reasonable to remove *all* the remaining explanatory variables from your best model so far? Fit a model with no explanatory variables, and do a test. (In R, if the right side of the squiggle is a `1`, that means "just an intercept". Or you can remove whatever remains using `update`.) What do you conclude? Explain briefly.

Solution

The fitting part is the challenge, since the testing part is `anova` again. The direct fit is this:

```
steak.3 = polr(steak_prep_ord ~ 1, data = steak)
```

and the `update` version is this, about equally long, starting from `steak.2` since that is the best model so far:

```
steak.3a = update(steak.2, . ~ . - educ - lottery_a)
```

You can use whichever you like. Either way, the second part is `anova`, and the two possible answers should be the same:

```
anova(steak.3, steak.2, test = "Chisq")
```

```
## Likelihood ratio tests of ordinal regression models
##
## Response: steak_prep_ord
##              Model Resid. df Resid. Dev
## 1                1       327   901.4467
## 2 educ + lottery_a       322   890.7028
##     Test    Df LR stat.    Pr(Chi)
## 1
## 2 1 vs 2     5 10.74387 0.05670146
```

or

```
anova(steak.3a, steak.2, test = "Chisq")
```

```
## Likelihood ratio tests of ordinal regression models
##
## Response: steak_prep_ord
##              Model Resid. df Resid. Dev
## 1                1       327   901.4467
## 2 educ + lottery_a       322   890.7028
##     Test    Df LR stat.    Pr(Chi)
## 1
## 2 1 vs 2     5 10.74387 0.05670146
```

At the 0.05 level, removing both of the remaining variables is fine: that is, nothing (out of these variables) has any impact on the probability that a diner will prefer their steak cooked a particular way.

However, with data like this and a rather exploratory analysis, I might think about using a larger α like 0.10, and at this level, taking out both these two variables is a bad idea. You could say that one or both of them is "potentially useful" or "provocative" or something like that.

If you think that removing these two variables is questionable, you might like to go back to that `drop1` output I had above:

```
drop1(steak.2, test = "Chisq")
```

```
## Single term deletions
##
## Model:
## steak_prep_ord ~ educ + lottery_a
##           Df    AIC    LRT Pr(>Chi)
## <none>       908.70
## educ       4 910.13 9.4299  0.05121 .
## lottery_a  1 907.96 1.2599  0.26167
## ---
```

```
## Signif. codes:
##   0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1  ' ' 1
```

The smallest AIC goes with `lottery_a`, so that comes out (it is nowhere near significant):

```
steak.4 = update(steak.2, . ~ . - lottery_a)
drop1(steak.4, test = "Chisq")
```

```
## Single term deletions
##
## Model:
## steak_prep_ord ~ educ
##        Df    AIC   LRT Pr(>Chi)
## <none>    907.96
## educ    4 909.45 9.484  0.05008 .
## ---
## Signif. codes:
##   0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1  ' ' 1
```

and what you see is that educational level is right on the edge of significance, so that may or may not have any impact. Make a call. But if anything, it's educational level that makes a difference.

(i) In the article for which these data were collected, link, does the author obtain consistent conclusions with yours? Explain briefly. (It's not a very long article, so it won't take you long to skim through, and the author's point is pretty clear.)

Solution

The article says that *nothing* has anything to do with steak preference. Whether you agree or not depends on what you thought above about dropping those last two variables. So say something consistent with what you said in the previous part. Two points for saying that the author said "nothing has any effect", and one point for how your findings square with that. Now that you have worked through this great long question, this is where I tell you that I simplified things a fair bit for you! There were lots of other variables that might have had an impact on how people like their steaks, and we didn't even consider those. Why did I choose what I did here? Well, I wanted to fit a regression predicting steak preference from everything else, do a big backward elimination, but:

```
steak.5 = polr(steak_prep_ord ~ ., data = steak)
```

```
## Warning: glm.fit: algorithm did not converge
```

```
## Error in polr(steak_prep_ord ~ ., data = steak): attempt to find suitable starting values failed
```

The . in place of explanatory variables means "all the other variables", including the nonsensical personal ID. That saved me having to type them all out.

Unfortunately, however, it didn't work. The problem is a numerical one. Regular regression has a well-defined procedure, where the computer follows through the steps and gets to the answer, every time. Once you go beyond regression, however, the answer is obtained by a step-by-step method: the computer makes an initial guess, tries to improve it, then tries to improve it again, until it can't improve things any more, at which point it calls it good. The problem here is that `polr` cannot even get the initial guess! (It apparently is known to suck at this, in problems as big and complicated as this one.)

I don't normally recommend forward selection, but I wonder whether it works here:

```
steak.5 = polr(steak_prep_ord ~ 1, data = steak)
steak.6 = step(steak.5, scope = . ~ lottery_a +
    smoke + alcohol + gamble + skydiving + speed +
    cheated + female + age + hhold_income + educ +
    region, direction = "forward", test = "Chisq",
    trace = 0)
drop1(steak.6, test = "Chisq")
```

```
## Single term deletions
##
## Model:
## steak_prep_ord ~ educ
##        Df    AIC   LRT Pr(>Chi)
## <none>    907.96
## educ    4 909.45 9.484  0.05008 .
## ---
## Signif. codes:
##   0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1  ' ' 1
```

It does, and it says the *only* thing to add out of all the variables is education level. So, for you, I picked this along with a couple of other plausible-sounding variables and had you start from there.

Forward selection starts from a model containing nothing and asks "what can we add?". This is a bit more complicated than backward elimination, because now you have to say what the candidate things to add *are*. That's the purpose of that `scope` piece, and there I had no alternative but to type the names of all the variables. Backward elimination is easier, because the candidate variables to remove are the ones in the model, and you don't need a `scope`. The `trace=0` says

"don't give me any output" (you can change it to a different value if you want to see what that does), and last, the `drop1` looks at what is actually *in* the final model (with a view to asking what can be removed, but we don't care about *that* here).

17.9 Crimes in San Francisco

The data in link is a subset of a huge dataset of crimes committed in San Francisco between 2003 and 2015. The variables are:

- `Dates`: the date and time of the crime
- `Category`: the category of crime, eg. "larceny" or "vandalism" (response).
- `Descript`: detailed description of crime.
- `DayOfWeek`: the day of the week of the crime.
- `PdDistrict`: the name of the San Francisco Police Department district in which the crime took place.
- `Resolution`: how the crime was resolved
- `Address`: approximate street address of crime
- `X`: longitude
- `Y`: latitude

Our aim is to see whether the category of crime depends on the day of the week and the district in which it occurred. However, there are a lot of crime categories, so we will focus on the top four "interesting" ones, which are the ones included in this data file.

Some of the model-fitting takes a while (you'll see why below). If you're using R Markdown, you can wait for the models to fit each time you re-run your document, or, edit the top line of a code chunk to say

$$````r, cache = T`$$

What that does is to re-run that code chunk only if it changes; if it hasn't changed it will use the saved results from last time it was run. That can save you a lot of waiting around.

(a) Read in the data and display the dataset (or, at least, part of it).

Solution

The usual:

```
my_url = "http://www.utsc.utoronto.ca/~butler/d29/sfcrime1.csv"
sfcrime = read_csv(my_url)
```

```
## Parsed with column specification:
## cols(
##   Category = col_character(),
##   DayOfWeek = col_character(),
##   PdDistrict = col_character()
## )
```

```
sfcrime
```

```
## # A tibble: 359,528 x 3
##    Category      DayOfWeek PdDistrict
##    <chr>         <chr>     <chr>
##  1 LARCENY/THEFT Wednesday NORTHERN
##  2 LARCENY/THEFT Wednesday PARK
##  3 LARCENY/THEFT Wednesday INGLESIDE
##  4 VEHICLE THEFT Wednesday INGLESIDE
##  5 VEHICLE THEFT Wednesday BAYVIEW
##  6 LARCENY/THEFT Wednesday RICHMOND
##  7 LARCENY/THEFT Wednesday CENTRAL
##  8 LARCENY/THEFT Wednesday CENTRAL
##  9 LARCENY/THEFT Wednesday NORTHERN
## 10 ASSAULT       Wednesday INGLESIDE
## # ... with 359,518 more rows
```

This is a tidied-up version of the data, with only the variables we'll look at, and only the observations from one of the "big four" crimes, a mere 300,000 of them. This is the data set we created earlier.

(b) Fit a multinomial logistic regression that predicts crime category from day of week and district. (You don't need to look at it.) The model-fitting produces some output. Hand that in too. (If you're using R Markdown, that will come with it.)

Solution

The modelling part is easy enough, as long as you can get the uppercase letters in the right places:

```
sfcrime.1 = multinom(Category ~ DayOfWeek + PdDistrict,
    data = sfcrime)
```

```
## # weights:  68 (48 variable)
## initial  value 498411.639069
## iter  10 value 430758.073422
## iter  20 value 430314.270403
```

```
## iter  30 value 423303.587698
## iter  40 value 420883.528523
## iter  50 value 418355.242764
## final  value 418149.979622
## converged
```

(c) Fit a model that predicts Category from only the district. Hand in the output from the fitting process as well.

Solution

Same idea. Write it out, or use `update`:

```
sfcrime.2 = update(sfcrime.1, . ~ . - DayOfWeek)
```

```
## # weights:  44 (30 variable)
## initial  value 498411.639069
## iter  10 value 426003.543845
## iter  20 value 425542.806828
## iter  30 value 421715.787609
## final  value 418858.235297
## converged
```

(d) Use `anova` to compare the two models you just obtained. What does the `anova` tell you?

Solution

This:

```
anova(sfcrime.2, sfcrime.1)
```

```
## Likelihood ratio tests of Multinomial Models
##
## Response: Category
##                    Model Resid. df
## 1             PdDistrict   1078554
## 2 DayOfWeek + PdDistrict   1078536
##   Resid. Dev   Test    Df LR stat. Pr(Chi)
## 1   837716.5
## 2   836300.0 1 vs 2    18 1416.511       0
```

This is a very small P-value. The null hypothesis is that the two models are equally good, and this is clearly rejected. We need the bigger model: that is, we need to keep `DayOfWeek` in there, because the pattern of crimes (in each district) differs over day of week.

One reason the P-value came out so small is that we have a ton of data, so that even a very small difference between days of the week could come out very strongly significant. The Machine Learning

people (this is a machine learning dataset) don't worry so much about tests for that reason: they are more concerned with predicting things well, so they just throw everything into the model and see what comes out.

(e) Using your preferred model, obtain predicted probabilities that a crime will be of each of these four categories for each day of the week in the `TENDERLOIN` district (the name is ALL CAPS). This will mean constructing a data frame to predict from, obtaining the predictions and then displaying them suitably.

Solution

I left this one fairly open, because you've done this kind of thing before, so what you need to do ought to be fairly clear: Construct the values to predict for with plural names:

You are almost certainly going to get the Capital Letters wrong in *DayOfWeek*, once, somewhere in the process. I did.

```
DayOfWeeks = sfcrime %>% distinct(DayOfWeek) %>%
    pull(DayOfWeek)
DayOfWeeks
```

```
## [1] "Wednesday" "Tuesday"   "Monday"
## [4] "Sunday"    "Saturday"  "Friday"
## [7] "Thursday"
```

```
PdDistricts = "TENDERLOIN"
```

The days of the week are in the order they came out in the data. I could fix this up, but the thing that matters for us is that Saturday and Sunday came out next to each other.

Anyway, it's about right, even though only one of the variables is actually plural and the other plural came out wrong.

Then, make data frame of combinations. The nice thing is that if we had more than one district, we'd just define `PdDistricts` appropriately above and the rest of it would be the same:

```
sfcrime.new = crossing(DayOfWeek = DayOfWeeks,
    PdDistrict = PdDistricts)
sfcrime.new
```

```
## # A tibble: 7 x 2
##   DayOfWeek PdDistrict
##   <chr>     <chr>
## 1 Friday    TENDERLOIN
## 2 Monday    TENDERLOIN
## 3 Saturday  TENDERLOIN
## 4 Sunday    TENDERLOIN
## 5 Thursday  TENDERLOIN
```

```
## 6 Tuesday   TENDERLOIN
## 7 Wednesday TENDERLOIN
```

Then do the predictions:

```
p = predict(sfcrime.1, sfcrime.new, type = "probs")
```

I can never remember the `type` thing for these models, because they are *all different*. I just make a guess, and if I guess something that this version of `predict` doesn't know about, it'll tell me:

```
p = predict(sfcrime.1, sfcrime.new, type = "bananas")
```

```
## Error in match.arg(type): 'arg' should be one of "class", "probs"
```

I can work out from this that `type` should be `probs`, since the other one makes the best guess at which category of response you'll get (the one with the highest probability). The predicted probabilities are more informative, since then you can see how they change, even if the predicted category stays the same.

This is something we'll see again in discriminant analysis.

Finally, display the results. I thought `cbind` wouldn't work here, because some of the variables are factors and some are numbers, but `cbind` is smarter than that:

```
cbind(sfcrime.new, p)
```

```
##   DayOfWeek PdDistrict   ASSAULT
## 1    Friday TENDERLOIN 0.2125997
## 2    Monday TENDERLOIN 0.2072430
## 3  Saturday TENDERLOIN 0.2425998
## 4    Sunday TENDERLOIN 0.2548597
## 5  Thursday TENDERLOIN 0.1938754
## 6   Tuesday TENDERLOIN 0.1942447
## 7 Wednesday TENDERLOIN 0.1874867
##   DRUG/NARCOTIC LARCENY/THEFT VEHICLE THEFT
## 1     0.4658122     0.2907793    0.03080879
## 2     0.5001466     0.2656069    0.02700348
## 3     0.4200751     0.3060611    0.03126405
## 4     0.4287291     0.2868682    0.02954299
## 5     0.5179500     0.2617658    0.02640888
## 6     0.5208668     0.2593331    0.02555535
## 7     0.5400287     0.2479498    0.02453490
```

(f) Describe briefly how the weekend days Saturday and Sunday differ from the rest.

Solution

The days ended up in some quasi-random order, but Saturday and Sunday are still together, so we can still easily compare them with the rest. My take is that the last two columns don't differ much between weekday and weekend, but the first two columns do: the probability of a crime being an assault is a bit higher on the weekend, and the probability of a crime being drug-related is a bit lower. I will accept anything reasonable supported by the predictions you got. We said there was a strongly significant day-of-week effect, but the changes from weekday to weekend are actually pretty small (but the changes from one weekday to another are even smaller). This supports what I guessed before, that with this much data even a small effect (the one shown here) is statistically significant. I want to compare another district. What districts do we have?

Statistical significance as an idea grew up in the days before *big data*.

```
sfcrime %>% count(PdDistrict)
```

```
## # A tibble: 10 x 2
##    PdDistrict     n
##    <chr>      <int>
##  1 BAYVIEW    31693
##  2 CENTRAL    38052
##  3 INGLESIDE  30102
##  4 MISSION    45277
##  5 NORTHERN   47750
##  6 PARK       19197
##  7 RICHMOND   18211
##  8 SOUTHERN   67981
##  9 TARAVAL    24981
## 10 TENDERLOIN 36284
```

This is the number of our "big four" crimes committed in each district. Let's look at the lowest-crime district `RICHMOND`. I copy and paste my code. Since I want to compare two districts, I include both:

```
DayOfWeeks = sfcrime %>% distinct(DayOfWeek) %>% pull(DayOfWeek)
PdDistricts = c("RICHMOND", "TENDERLOIN")
sfcrime.new = crossing(DayOfWeek = DayOfWeeks, PdDistrict = PdDistricts)
sfcrime.new
```

```
## # A tibble: 14 x 2
##    DayOfWeek PdDistrict
##    <chr>     <chr>
##  1 Friday    RICHMOND
##  2 Friday    TENDERLOIN
##  3 Monday    RICHMOND
##  4 Monday    TENDERLOIN
```

```
##  5 Saturday  RICHMOND
##  6 Saturday  TENDERLOIN
##  7 Sunday    RICHMOND
##  8 Sunday    TENDERLOIN
##  9 Thursday  RICHMOND
## 10 Thursday  TENDERLOIN
## 11 Tuesday   RICHMOND
## 12 Tuesday   TENDERLOIN
## 13 Wednesday RICHMOND
## 14 Wednesday TENDERLOIN
```

```
p1 = predict(sfcrime.1, sfcrime.new, type = "probs")
d1 = cbind(sfcrime.new, p1)
d1
```

```
##    DayOfWeek PdDistrict   ASSAULT DRUG/NARCOTIC LARCENY/THEFT VEHICLE THEFT
## 1     Friday   RICHMOND 0.1666304    0.04944552     0.5466635    0.23726062
## 2     Friday TENDERLOIN 0.2125997    0.46581221     0.2907793    0.03080879
## 3     Monday   RICHMOND 0.1760175    0.05753044     0.5411034    0.22534869
## 4     Monday TENDERLOIN 0.2072430    0.50014661     0.2656069    0.02700348
## 5   Saturday   RICHMOND 0.1809352    0.04243108     0.5475273    0.22910644
## 6   Saturday TENDERLOIN 0.2425998    0.42007509     0.3060611    0.03126405
## 7     Sunday   RICHMOND 0.1973675    0.04496576     0.5328708    0.22479591
## 8     Sunday TENDERLOIN 0.2548597    0.42872910     0.2868682    0.02954299
## 9   Thursday   RICHMOND 0.1683841    0.06092432     0.5453260    0.22536564
## 10  Thursday TENDERLOIN 0.1938754    0.51795000     0.2617658    0.02640888
## 11   Tuesday   RICHMOND 0.1706999    0.06199196     0.5466472    0.22066087
## 12   Tuesday TENDERLOIN 0.1942447    0.52086682     0.2593331    0.02555535
## 13 Wednesday   RICHMOND 0.1709963    0.06670490     0.5424317    0.21986703
## 14 Wednesday TENDERLOIN 0.1874867    0.54002868     0.2479498    0.02453490
```

Richmond is obviously not a drug-dealing kind of place; most of its crimes are theft of one kind or another. But the predicted effect of weekday vs. weekend is the same: Richmond doesn't have many assaults or drug crimes, but it also has more assaults and fewer drug crimes on the weekend than during the week. There is not much effect of day of the week on the other two crime types in either place.

The consistency of *pattern*, even though the prevalence of the different crime types differs by location, is a feature of the model: we fitted an additive model, that says there is an effect of weekday, and *independently* there is an effect of location. The *pattern* over weekday is the same for each location, implied by the model. This may or may not be supported by the actual data.

The way to assess this is to fit a model with interaction (we will see more of this when we revisit ANOVA later), and compare the fit:

```
sfcrime.3 = update(sfcrime.1, . ~ . + DayOfWeek * PdDistrict)
```

```
## # weights:  284 (210 variable)
## initial  value 498411.639069
## iter  10 value 429631.807781
## iter  20 value 429261.427210
## iter  30 value 428111.625547
## iter  40 value 423807.031450
## iter  50 value 421129.496196
## iter  60 value 420475.833895
## iter  70 value 419523.235916
## iter  80 value 418621.612920
## iter  90 value 418147.629782
## iter 100 value 418036.670485
## final  value 418036.670485
## stopped after 100 iterations
```

```
# anova(sfcrime.1,sfcrime.3)
```

This one didn't actually complete the fitting process: it got to 100 times around and stopped (since that's the default limit). We can make it go a bit further thus:

```
sfcrime.3 = update(sfcrime.1, . ~ . + DayOfWeek * PdDistrict, maxit = 300)
```

```
## # weights:  284 (210 variable)
## initial  value 498411.639069
## iter  10 value 429631.807781
## iter  20 value 429261.427210
## iter  30 value 428111.625547
## iter  40 value 423807.031450
## iter  50 value 421129.496196
## iter  60 value 420475.833895
## iter  70 value 419523.235916
## iter  80 value 418621.612920
## iter  90 value 418147.629782
## iter 100 value 418036.670485
## iter 110 value 417957.337016
## iter 120 value 417908.465189
## iter 130 value 417890.580843
## iter 140 value 417874.839492
## iter 150 value 417867.449343
## iter 160 value 417862.626213
## iter 170 value 417858.654628
## final  value 417858.031855
## converged
```

```r
anova(sfcrime.1, sfcrime.3)
```

```
## Likelihood ratio tests of Multinomial Models
##
## Response: Category
##                                          Model Resid. df Resid. Dev   Test    Df
## 1                        DayOfWeek + PdDistrict   1078536   836300.0
## 2 DayOfWeek + PdDistrict + DayOfWeek:PdDistrict   1078374   835716.1 1 vs 2   162
##   LR stat. Pr(Chi)
## 1
## 2 583.8955       0
```

This time, we got to the end. (The `maxit=300` gets passed on to `multinom`, and says "go around up to 300 times if needed".) As you will see if you try it, this takes a bit of time to run.

This `anova` is also strongly significant, but in the light of the previous discussion, the differential effect of day of week in different districts might not be very big. We can even assess that; we have all the machinery for the predictions, and we just have to apply them to this model:

```r
p3 = predict(sfcrime.3, sfcrime.new, type = "probs")
d3 = cbind(sfcrime.new, p3)
d3
```

```
##    DayOfWeek PdDistrict   ASSAULT DRUG/NARCOTIC LARCENY/THEFT VEHICLE THEFT
## 1     Friday   RICHMOND 0.1756062    0.05241934     0.5447797    0.22719474
## 2     Friday TENDERLOIN 0.2163513    0.44958359     0.3006968    0.03336831
## 3     Monday   RICHMOND 0.1668855    0.06038366     0.5537600    0.21897087
## 4     Monday TENDERLOIN 0.2061337    0.51256922     0.2569954    0.02430166
## 5   Saturday   RICHMOND 0.1859514    0.04414673     0.5244262    0.24547574
## 6   Saturday TENDERLOIN 0.2368561    0.41026767     0.3221820    0.03069417
## 7     Sunday   RICHMOND 0.1851406    0.05211970     0.5403967    0.22234300
## 8     Sunday TENDERLOIN 0.2570331    0.40019234     0.3101184    0.03265625
## 9   Thursday   RICHMOND 0.1811185    0.05621201     0.5394484    0.22322107
## 10  Thursday TENDERLOIN 0.1951851    0.52867900     0.2528465    0.02328943
## 11   Tuesday   RICHMOND 0.1758111    0.05587178     0.5471405    0.22117663
## 12   Tuesday TENDERLOIN 0.1907137    0.53955543     0.2419757    0.02775520
## 13 Wednesday   RICHMOND 0.1603796    0.06279956     0.5525565    0.22426437
## 14 Wednesday TENDERLOIN 0.1906309    0.54577677     0.2401585    0.02343381
```

It doesn't *look* much different. Maybe the Tenderloin has a larger weekend increase in assaults than Richmond does. I saved these predictions in data frames so that I could compare them. Columns 3 through 6 contain the actual predictions; let's take the difference between them, and glue it back onto the district and day of week.

I rounded the prediction differences off to 4 decimals to make the largest ones easier to find:

```
pdiff = round(p3 - p1, 4)
cbind(sfcrime.new, pdiff)
```

```
##    DayOfWeek PdDistrict ASSAULT DRUG/NARCOTIC LARCENY/THEFT VEHICLE THEFT
## 1     Friday   RICHMOND  0.0090        0.0030       -0.0019       -0.0101
## 2     Friday TENDERLOIN  0.0038       -0.0162        0.0099        0.0026
## 3     Monday   RICHMOND -0.0091        0.0029        0.0127       -0.0064
## 4     Monday TENDERLOIN -0.0011        0.0124       -0.0086       -0.0027
## 5   Saturday   RICHMOND  0.0050        0.0017       -0.0231        0.0164
## 6   Saturday TENDERLOIN -0.0057       -0.0098        0.0161       -0.0006
## 7     Sunday   RICHMOND -0.0122        0.0072        0.0075       -0.0025
## 8     Sunday TENDERLOIN  0.0022       -0.0285        0.0233        0.0031
## 9   Thursday   RICHMOND  0.0127       -0.0047       -0.0059       -0.0021
## 10  Thursday TENDERLOIN  0.0013        0.0107       -0.0089       -0.0031
## 11   Tuesday   RICHMOND  0.0051       -0.0061        0.0005        0.0005
## 12   Tuesday TENDERLOIN -0.0035        0.0187       -0.0174        0.0022
## 13 Wednesday   RICHMOND -0.0106       -0.0039        0.0101        0.0044
## 14 Wednesday TENDERLOIN  0.0031        0.0057       -0.0078       -0.0011
```

None of the predicted probabilities differ by more than 0.04, which is consistent with the size of the interaction effect being small even though significant. Some of the differences largest in size are Drugs-Narcotics in Tenderloin, but even they are not big.

The programmer in me wants to find a way to display the largest and smallest few differences. My idea is to use `gather` to make *one* column of differences and sort it, then display the top and bottom few values. Note that I have to put those little "backticks" around `VEHICLE THEFT` since the variable name has a space in it. My code inefficiently sorts the differences twice:

```
d1 = cbind(sfcrime.new, pdiff) %>% gather(crimetype, difference, ASSAULT:`VEHICLE THEFT`)
d1 %>% arrange(difference) %>% slice(1:6)
```

```
##   DayOfWeek PdDistrict     crimetype difference
## 1    Sunday TENDERLOIN DRUG/NARCOTIC    -0.0285
## 2  Saturday   RICHMOND LARCENY/THEFT    -0.0231
## 3   Tuesday TENDERLOIN LARCENY/THEFT    -0.0174
## 4    Friday TENDERLOIN DRUG/NARCOTIC    -0.0162
## 5    Sunday   RICHMOND       ASSAULT    -0.0122
## 6 Wednesday   RICHMOND       ASSAULT    -0.0106
```

```
d1 %>% arrange(desc(difference)) %>% slice(1:6)
```

```
##   DayOfWeek PdDistrict     crimetype difference
## 1    Sunday TENDERLOIN LARCENY/THEFT     0.0233
## 2   Tuesday TENDERLOIN DRUG/NARCOTIC     0.0187
## 3  Saturday   RICHMOND VEHICLE THEFT     0.0164
## 4  Saturday TENDERLOIN LARCENY/THEFT     0.0161
## 5  Thursday   RICHMOND       ASSAULT     0.0127
## 6    Monday   RICHMOND LARCENY/THEFT     0.0127
```

I don't know what, if anything, you make of those.

Extra: there is a better way of doing those in one go:

```
d1 %>% top_n(6, difference)
```

```
##   DayOfWeek PdDistrict     crimetype difference
## 1  Thursday   RICHMOND       ASSAULT     0.0127
## 2   Tuesday TENDERLOIN DRUG/NARCOTIC     0.0187
## 3    Monday   RICHMOND LARCENY/THEFT     0.0127
## 4  Saturday TENDERLOIN LARCENY/THEFT     0.0161
## 5    Sunday TENDERLOIN LARCENY/THEFT     0.0233
## 6  Saturday   RICHMOND VEHICLE THEFT     0.0164
```

```
d1 %>% top_n(6, -difference)
```

```
##   DayOfWeek PdDistrict     crimetype difference
## 1    Sunday   RICHMOND       ASSAULT    -0.0122
## 2 Wednesday   RICHMOND       ASSAULT    -0.0106
## 3    Friday TENDERLOIN DRUG/NARCOTIC    -0.0162
## 4    Sunday TENDERLOIN DRUG/NARCOTIC    -0.0285
## 5  Saturday   RICHMOND LARCENY/THEFT    -0.0231
## 6   Tuesday TENDERLOIN LARCENY/THEFT    -0.0174
```

that is just slicker. What it actually does is almost the same as my code, but it saves you worrying about the details. (It picks out the top 6, but without putting them in order.)

17.10 *High School and Beyond*

A survey called High School and Beyond was given to a large number of American high school seniors (grade 12) by the National Center of Education Statistics. The data set at link is a random sample of 200 of those students.

The variables collected are:

- `gender`: student's gender, female or male.
- `race`: the student's race (African-American, Asian, Hispanic, White).

I'm always amused at how Americans put all Asians into one group.

- `ses`: Socio-economic status of student's family (low, middle, or high)
- `schtyp`: School type, public or private.
- `prog`: Student's program, general, academic, or vocational.
- `read`: Score on standardized reading test.
- `write`: Score on standardized writing test.
- `math`: Score on standardized math test.
- `science`: Score on standardized science test.
- `socst`: Score on standardized social studies test.

Our aim is to see how socio-economic status is related to the other variables.

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

Solution
This is a `.csv` file (I tried to make it easy for you):

```
my_url = "http://www.utsc.utoronto.ca/~butler/d29/hsb.csv"
hsb = read_csv(my_url)
```

```
## Parsed with column specification:
## cols(
##   id = col_integer(),
##   race = col_character(),
##   ses = col_character(),
##   schtyp = col_character(),
##   prog = col_character(),
##   read = col_integer(),
##   write = col_integer(),
##   math = col_integer(),
##   science = col_integer(),
##   socst = col_integer(),
##   gender = col_character()
## )
```

```
hsb
```

```
## # A tibble: 200 x 11
##       id race         ses    schtyp prog        read write  math science socst gender
##    <int> <chr>        <chr>  <chr>  <chr>      <int> <int> <int>   <int> <int> <chr>
##  1    70 white        low    public general       57    52    41      47    57 male
```

```
##  2   121 white        middle public vocational    68    59    53      63    61 female
##  3    86 white        high   public general       44    33    54      58    31 male
##  4   141 white        high   public vocational    63    44    47      53    56 male
##  5   172 white        middle public academic      47    52    57      53    61 male
##  6   113 white        middle public academic      44    52    51      63    61 male
##  7    50 african-amer middle public general       50    59    42      53    61 male
##  8    11 hispanic     middle public academic      34    46    45      39    36 male
##  9    84 white        middle public general       63    57    54      58    51 male
## 10    48 african-amer middle public academic      57    55    52      50    51 male
## # ... with 190 more rows
```

(b) Explain briefly why an ordinal logistic regression is appropriate for our aims.

Solution

The response variable `ses` is categorical, with categories that come in order (low less than middle less than high).

(c) Fit an ordinal logistic regression predicting socio-economic status from the scores on the five standardized tests. (You don't need to display the results.) You will probably go wrong the first time. What kind of thing does your response variable have to be?

Solution

It has to be an `ordered` factor, which you can create in the data frame (or outside, if you prefer):

```
hsb = hsb %>% mutate(ses = ordered(ses, c("low", "middle", "high")))
hsb
```

```
## # A tibble: 200 x 11
##       id race         ses    schtyp prog        read write  math science socst gender
##    <int> <chr>        <ord>  <chr>  <chr>      <int> <int> <int>   <int> <int> <chr>
##  1    70 white        low    public general       57    52    41      47    57 male
##  2   121 white        middle public vocational    68    59    53      63    61 female
##  3    86 white        high   public general       44    33    54      58    31 male
##  4   141 white        high   public vocational    63    44    47      53    56 male
##  5   172 white        middle public academic      47    52    57      53    61 male
##  6   113 white        middle public academic      44    52    51      63    61 male
##  7    50 african-amer middle public general       50    59    42      53    61 male
##  8    11 hispanic     middle public academic      34    46    45      39    36 male
##  9    84 white        middle public general       63    57    54      58    51 male
## 10    48 african-amer middle public academic      57    55    52      50    51 male
## # ... with 190 more rows
```

`ses` is now `ord`. Good. Now fit the model:

```
ses.1 = polr(ses ~ read + write + math + science + socst, data = hsb)
```

No errors is good.

(d) Remove any non-significant explanatory variables one at a time. Use `drop1` to decide which one to remove next.

Solution

```
drop1(ses.1, test = "Chisq")
```

```
## Single term deletions
##
## Model:
## ses ~ read + write + math + science + socst
##         Df    AIC    LRT Pr(>Chi)
## <none>     404.63
## read     1 403.09 0.4620 0.496684
## write    1 403.81 1.1859 0.276167
## math     1 403.19 0.5618 0.453517
## science  1 404.89 2.2630 0.132499
## socst    1 410.08 7.4484 0.006349 **
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

I would have expected the AIC column to come out in order, but it doesn't. Never mind. Scan for the largest P-value, which belongs to `read`. (This also has the lowest AIC.) So, remove `read`:

```
ses.2 = update(ses.1, . ~ . - read)
drop1(ses.2, test = "Chisq")
```

```
## Single term deletions
##
## Model:
## ses ~ write + math + science + socst
##         Df    AIC    LRT Pr(>Chi)
## <none>     403.09
## write    1 402.10 1.0124 0.314325
## math     1 402.04 0.9541 0.328689
## science  1 404.29 3.1968 0.073782 .
## socst    1 410.58 9.4856 0.002071 **
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Note how the P-value for `science` has come down a long way.

A close call, but `math` goes next. The `update` doesn't take long to type:

```
ses.3 = update(ses.2, . ~ . - math)
drop1(ses.3, test = "Chisq")
```

```
## Single term deletions
##
## Model:
## ses ~ write + science + socst
##         Df    AIC     LRT  Pr(>Chi)
## <none>     402.04
## write    1 400.60  0.5587 0.4547813
## science  1 405.41  5.3680 0.0205095 *
## socst    1 411.07 11.0235 0.0008997 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

`science` has become significant now (probably because it was strongly correlated with at least one of the variables we removed (at my guess, `math`). That is, we didn't need both `science` and `math`, but we *do* need *one* of them.

I think we can guess what will happen now: `write` comes out, and the other two variables will stay, so that'll be where we stop:

```
ses.4 = update(ses.3, . ~ . - write)
drop1(ses.4, test = "Chisq")
```

```
## Single term deletions
##
## Model:
## ses ~ science + socst
##         Df    AIC     LRT  Pr(>Chi)
## <none>     400.60
## science  1 403.45  4.8511 0.0276291 *
## socst    1 409.74 11.1412 0.0008443 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Indeed so. We need just the science and social studies test scores to predict socio-economic status.

Using AIC to decide on which variable to remove next will give the same answer here, but I would like to see the `test=` part in your `drop1` to give P-values (expect to lose something, but not too much, if that's not there).

Extras: I talked about correlation among the explanatory variables earlier, which I can explore:

```
hsb %>% dplyr::select(read:socst) %>% cor()
```

```
##              read     write      math   science     socst
## read    1.0000000 0.5967765 0.6622801 0.6301579 0.6214843
## write   0.5967765 1.0000000 0.6174493 0.5704416 0.6047932
## math    0.6622801 0.6174493 1.0000000 0.6307332 0.5444803
## science 0.6301579 0.5704416 0.6307332 1.0000000 0.4651060
## socst   0.6214843 0.6047932 0.5444803 0.4651060 1.0000000
```

The first time I did this, I forgot that I had `MASS` loaded (for the `polr`), and so, to get the right `select`, I needed to say which one I wanted.

Anyway, the correlations are all moderately high. There's nothing that stands out as being much higher than the others. The lowest two are between social studies and math, and social studies and science. That would be part of the reason that social studies needs to stay. The highest correlation is between math and reading, which surprises me (they seem to be different skills).

So there was not as much insight there as I expected.

The other thing is that you can use `step` for the variable-elimination task as well:

```
ses.5 = step(ses.1, direction = "backward", test = "Chisq")
```

```
## Start:  AIC=404.63
## ses ~ read + write + math + science + socst
##
##           Df    AIC    LRT Pr(>Chi)
## - read     1 403.09 0.4620 0.496684
## - math     1 403.19 0.5618 0.453517
## - write    1 403.81 1.1859 0.276167
## <none>       404.63
## - science  1 404.89 2.2630 0.132499
## - socst    1 410.08 7.4484 0.006349 **
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Step:  AIC=403.09
## ses ~ write + math + science + socst
##
##           Df    AIC    LRT Pr(>Chi)
## - math     1 402.04 0.9541 0.328689
## - write    1 402.10 1.0124 0.314325
## <none>       403.09
## - science  1 404.29 3.1968 0.073782 .
## - socst    1 410.58 9.4856 0.002071 **
## ---
```

```
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Step:  AIC=402.04
## ses ~ write + science + socst
##
##           Df    AIC     LRT  Pr(>Chi)
## - write    1 400.60  0.5587 0.4547813
## <none>       402.04
## - science  1 405.41  5.3680 0.0205095 *
## - socst    1 411.07 11.0235 0.0008997 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Step:  AIC=400.6
## ses ~ science + socst
##
##           Df    AIC     LRT  Pr(>Chi)
## <none>       400.60
## - science  1 403.45  4.8511 0.0276291 *
## - socst    1 409.74 11.1412 0.0008443 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

I would accept you doing it this way, again *as long as you have the 'test=' there as well.*

(e) The quartiles of the `science` test score are 44 and 58. The quartiles of the `socst` test score are 46 and 61. Make a data frame that has all combinations of those quartiles. If your best regression had any other explanatory variables in it, also put the *medians* of those variables into this data frame.

Solution

Thus, most obviously:

```r
sciences = c(44, 58)
socsts = c(46, 61)
new = crossing(science = sciences, socst = socsts)
new
```

```
## # A tibble: 4 x 2
##   science socst
##     <dbl> <dbl>
## 1      44    46
## 2      44    61
## 3      58    46
## 4      58    61
```

Since there are only two variables left, this new data frame has only $2^2 = 4$ rows.

There is a veiled hint here that these are the two variables that should have remained in your regression. If that was not what you got, find the median of any other variables you had, and put that into your `new`. For example, if you still had `math`, you'd do this:

That is *maths* as the apparently-plural of *math*, not as the British name for mathematics.

```
hsb %>% summarize(m = median(math))
```

```
## # A tibble: 1 x 1
##       m
##   <dbl>
## 1    52
```

```
maths = 52
new2 = crossing(science = sciences, socst = socsts, math = maths)
new2
```

```
## # A tibble: 4 x 3
##   science socst  math
##     <dbl> <dbl> <dbl>
## 1      44    46    52
## 2      44    61    52
## 3      58    46    52
## 4      58    61    52
```

(f) Use the data frame you created in the previous part, together with your best model, to obtain predicted probabilities of being in each `ses` category.

Solution

This is `predict`, and we've done the setup. My best model was called `ses.4`:

```
p = predict(ses.4, new, type = "probs")
cbind(new, p)
```

```
##   science socst       low    middle      high
## 1      44    46 0.3262510 0.5056113 0.1681377
## 2      44    61 0.1885566 0.5150763 0.2963671
## 3      58    46 0.2278105 0.5230783 0.2491112
## 4      58    61 0.1240155 0.4672338 0.4087507
```

If you forgot which `type` to use, guess one that's obviously wrong, and it will tell you what your options are. That ought to be enough to trigger your memory of the right thing.

If you get the `type` right, but the `predict` still doesn't work, check the construction of your `new` data frame very carefully. That's likely where your problem is.

(g) What is the effect of an increased science score on the likelihood of a student being in the different socioeconomic groups, all else equal? Explain briefly. In your explanation, state clearly how you are using your answer to the previous part.

Solution

Use your predictions; hold the `socst` score constant (that's the all else equal part). So compare the first and third rows (or, if you like, the second and fourth rows) of your predictions and see what happens as the science score goes from 44 to 58. What I see is that the probability of being `low` goes noticeably *down* as the science score increases, the probability of `middle` stays about the same, and the probability of `high` goes `up` (by about the same amount as the probability of `low` went down). In other words, an increased science score goes with an increased chance of `high` (and a decreased chance of `low`). If your best model doesn't have `science` in it, then you need to say something like "`science` has no effect on socio-economic status", consistent with what you concluded before: if you took it out, it's because you thought it had no effect. Extra: the effect of an increased social studies score is almost exactly the same as an increased science score (so I didn't ask you about that). From a social-science point of view, this makes perfect sense: the higher the social-economic stratum a student comes from, the better they are likely to do in school. I've been phrasing this as "association", because really the cause and effect is the other way around: a student's family socioeconomic status is explanatory, and school performance is response. But this was the nicest example I could find of an ordinal response data set.

17.11 *What sports do these athletes play?*

The data at link are physical and physiological measurements of 202 male and female Australian elite athletes. The data values are separated by *tabs*. We are going to see whether we can predict the sport an athlete plays from their height and weight.

The sports, if you care, are respectively basketball, "field athletics" (eg. shot put, javelin throw, long jump etc.), gymnastics, netball, rowing, swimming, 400m running, tennis, sprinting (100m or 200m running), water polo.

(a) Read in the data and display the first few rows.

Solution

The data values are separated by tabs, so `read_tsv` is the thing:

```
my_url = "http://www.utsc.utoronto.ca/~butler/c32/ais.txt"
athletes = read_tsv(my_url)
```

```
## Parsed with column specification:
## cols(
##   Sex = col_character(),
##   Sport = col_character(),
##   RCC = col_double(),
##   WCC = col_double(),
##   Hc = col_double(),
##   Hg = col_double(),
##   Ferr = col_integer(),
##   BMI = col_double(),
##   SSF = col_double(),
##   `%Bfat` = col_double(),
##   LBM = col_double(),
##   Ht = col_double(),
##   Wt = col_double()
## )
```

```
athletes
```

```
## # A tibble: 202 x 13
##    Sex    Sport     RCC   WCC    Hc    Hg  Ferr   BMI   SSF `%Bfat`   LBM    Ht    Wt
##    <chr>  <chr>   <dbl> <dbl> <dbl> <dbl> <int> <dbl> <dbl>   <dbl> <dbl> <dbl> <dbl>
##  1 female Netball  4.56  13.3  42.2  13.6    20  19.2  49      11.3  53.1  177.  59.9
##  2 female Netball  4.15   6    38    12.7    59  21.2 110.     25.3  47.1  173.  63
##  3 female Netball  4.16   7.6  37.5  12.3    22  21.4  89      19.4  53.4  176   66.3
##  4 female Netball  4.32   6.4  37.7  12.3    30  21.0  98.3    19.6  48.8  170.  60.7
##  5 female Netball  4.06   5.8  38.7  12.8    78  21.8 122.     23.1  56.0  183   72.9
##  6 female Netball  4.12   6.1  36.6  11.8    21  21.4  90.4    16.9  56.4  178.  67.9
##  7 female Netball  4.17   5    37.4  12.7   109  21.5 107.     21.3  53.1  177.  67.5
##  8 female Netball  3.8    6.6  36.5  12.4   102  24.4 157.     26.6  54.4  174.  74.1
##  9 female Netball  3.96   5.5  36.3  12.4    71  22.6 101.     17.9  56.0  174.  68.2
## 10 female Netball  4.44   9.7  41.4  14.1    64  22.8 126.     25.0  51.6  174.  68.8
## # ... with 192 more rows
```

If you didn't remember that, this also works:

```
athletes = read_delim(my_url, "\t")
```

```
## Parsed with column specification:
## cols(
##   Sex = col_character(),
##   Sport = col_character(),
##   RCC = col_double(),
##   WCC = col_double(),
##   Hc = col_double(),
##   Hg = col_double(),
```

```
##   Ferr = col_integer(),
##   BMI = col_double(),
##   SSF = col_double(),
##   `%Bfat` = col_double(),
##   LBM = col_double(),
##   Ht = col_double(),
##   Wt = col_double()
## )
```

(this is the R way of expressing "tab".)

(b) Make a scatterplot of height vs. weight, with the points coloured by what sport the athlete plays. Put height on the x-axis and weight on the y-axis.

Solution

I'm doing this to give you a little intuition for later:

```
ggplot(athletes, aes(x = Ht, y = Wt, colour = Sport)) + geom_point()
```

The reason for giving you the axes to use is (i) neither variable is really a response, so it doesn't actually matter which one goes on which axis, and (ii) I wanted to give the grader something consistent to look at.

(c) Explain briefly why a multinomial model (`multinom` from `nnet`) would be the best thing to use to predict sport played from the other variables.

Solution

The categories of `Sport` are not in any kind of order, and there are more than two of them. That's really all you needed, for which two marks is kind of generous.

(d) Fit a suitable model for predicting sport played from height and weight. (You don't need to look at the results.) 100 steps isn't quite enough, so set `maxit` equal to a larger number to allow the estimation to finish.

Solution

120 steps is actually enough, but any number larger than 110 is fine. It doesn't matter if your guess is way too high. Like this:

```
library(nnet)
sport.1 = multinom(Sport ~ Ht + Wt, data = athletes, maxit = 200)
```

```
## # weights:  40 (27 variable)
## initial  value 465.122189
## iter  10 value 410.089598
## iter  20 value 391.426721
## iter  30 value 365.829150
## iter  40 value 355.326457
## iter  50 value 351.255395
## iter  60 value 350.876479
## iter  70 value 350.729699
## iter  80 value 350.532323
## iter  90 value 350.480130
## iter 100 value 350.349271
## iter 110 value 350.312029
## final  value 350.311949
## converged
```

As long as you see the word `converged` at the end, you're good.

(e) Demonstrate using `anova` that `Wt` should not be removed from this model.

Solution

The idea is to fit a model without `Wt`, and then show that it fits significantly worse:

```
sport.2 = update(sport.1, . ~ . - Wt)
```

```
## # weights:  30 (18 variable)
## initial  value 465.122189
## iter  10 value 447.375728
## iter  20 value 413.597441
## iter  30 value 396.685596
## iter  40 value 394.121380
## iter  50 value 394.116993
## iter  60 value 394.116434
```

```
## final  value 394.116429
## converged
```

```r
anova(sport.2, sport.1, test = "Chisq")
```

```
## Likelihood ratio tests of Multinomial Models
##
## Response: Sport
##     Model Resid. df Resid. Dev   Test    Df LR stat.      Pr(Chi)
## 1      Ht      1800   788.2329
## 2 Ht + Wt      1791   700.6239 1 vs 2     9 87.60896 4.884981e-15
```

The P-value is very small indeed, so the bigger model `sport.1` is definitely better (or the smaller model `sport.2` is significantly worse, however you want to say it). So taking `Wt` out is definitely a mistake.

This is what I would have guessed (I actually wrote the question in anticipation of this being the answer) because weight certainly seems to help in distinguishing the sports. For example, the field athletes seem to be heavy for their height compared to the other athletes (look back at the graph you made).

`drop1`, the obvious thing, doesn't work here:

```r
drop1(sport.1, test = "Chisq", trace = T)
```

```
## trying - Ht
```

```
## Error in if (trace) {: argument is not interpretable as logical
```

I gotta figure out what that error is. Does `step`?

```r
step(sport.1, direction = "backward", test = "Chisq")
```

```
## Start:  AIC=754.62
## Sport ~ Ht + Wt
##
## trying - Ht
## # weights:  30 (18 variable)
## initial  value 465.122189
## iter  10 value 441.367394
## iter  20 value 381.021649
## iter  30 value 380.326030
## final  value 380.305003
## converged
## trying - Wt
## # weights:  30 (18 variable)
## initial  value 465.122189
## iter  10 value 447.375728
```

```
## iter  20 value 413.597441
## iter  30 value 396.685596
## iter  40 value 394.121380
## iter  50 value 394.116993
## iter  60 value 394.116434
## final  value 394.116429
## converged
##        Df      AIC
## <none> 27 754.6239
## - Ht   18 796.6100
## - Wt   18 824.2329
```

```
## Call:
## multinom(formula = Sport ~ Ht + Wt, data = athletes, maxit = 200)
##
## Coefficients:
##         (Intercept)         Ht          Wt
## Field      59.98535 -0.4671650  0.31466413
## Gym       112.49889 -0.5027056 -0.57087657
## Netball    47.70209 -0.2947852  0.07992763
## Row        35.90829 -0.2517942  0.14164007
## Swim       36.82832 -0.2444077  0.10544986
## T400m      32.73554 -0.1482589 -0.07830622
## Tennis     41.92855 -0.2278949 -0.01979877
## TSprnt     51.43723 -0.3359534  0.12378285
## WPolo      23.35291 -0.2089807  0.18819526
##
## Residual Deviance: 700.6239
## AIC: 754.6239
```

Curiously enough, it does. The final model is the same as the initial one, telling us that neither variable should be removed.

(f) Make a data frame consisting of all combinations of `Ht` 160, 180 and 200 (cm), and `Wt` 50, 75, and 100 (kg), and use it to obtain predicted probabilities of athletes of those heights and weights playing each of the sports. Display the results. You might have to display them smaller, or reduce the number of decimal places to fit them on the page.

For this, use *round*.

Solution

This is (again) the easier way: fill vectors with the given values, use `crossing` to get the combinations, and feed that into `predict`, thus:

```
Hts = c(160, 180, 200)
Wts = c(50, 75, 100)
```

```
new = crossing(Ht = Hts, Wt = Wts)
new
```

```
## # A tibble: 9 x 2
##      Ht    Wt
##   <dbl> <dbl>
## 1   160    50
## 2   160    75
## 3   160   100
## 4   180    50
## 5   180    75
## 6   180   100
## 7   200    50
## 8   200    75
## 9   200   100
```

and then

```
p = predict(sport.1, new, type = "probs")
cbind(new, p)
```

```
##    Ht  Wt        BBall        Field          Gym      Netball          Row         Swim
## 1 160  50 2.147109e-03 5.676191e-03 7.269646e-02 0.1997276344 0.0320511810 0.0429354624
## 2 160  75 1.044581e-04 7.203893e-01 2.240720e-09 0.0716686494 0.0537984065 0.0291616004
## 3 160 100 5.541050e-08 9.968725e-01 7.530504e-19 0.0002804029 0.0009845934 0.0002159577
## 4 180  50 9.142760e-02 2.116103e-05 1.331347e-04 0.0233985667 0.0088717680 0.0137765580
## 5 180  75 7.787300e-02 4.701849e-02 7.184344e-11 0.1469948033 0.2607097148 0.1638164831
## 6 180 100 5.379479e-04 8.473139e-01 3.144321e-19 0.0074895973 0.0621366751 0.0157985864
## 7 200  50 6.907544e-01 1.399715e-08 4.326057e-08 0.0004863664 0.0004357120 0.0007843113
## 8 200  75 8.889269e-01 4.698989e-05 3.527126e-14 0.0046164597 0.0193454719 0.0140908812
## 9 200 100 2.738448e-01 3.776290e-02 6.884080e-21 0.0104894025 0.2056152576 0.0606015871
##          T400m       Tennis       TSprnt        WPolo
## 1 3.517558e-01 1.885847e-01 1.033315e-01 1.094045e-03
## 2 2.416185e-03 5.592833e-03 1.109879e-01 5.880674e-03
## 3 1.809594e-07 1.808504e-06 1.299813e-03 3.446523e-04
## 4 7.721547e-01 8.418976e-02 5.313751e-03 7.129763e-04
## 5 9.285706e-02 4.371258e-02 9.992310e-02 6.709478e-02
## 6 9.056685e-05 1.840761e-04 1.523963e-02 5.120898e-02
## 7 3.007396e-01 6.668610e-03 4.848339e-05 8.244005e-05
## 8 5.464293e-02 5.231368e-03 1.377496e-03 1.172154e-02
## 9 2.376695e-03 9.824069e-04 9.368804e-03 3.989581e-01
```

I'll take this, but read on for an improvement.

This still spills onto a second line (even printed this small). Let's round the predicted probabilities to 2 decimals, which, with luck, will also kill the scientific notation:

```r
cbind(new, round(p, 2))
```

```
##    Ht  Wt BBall Field  Gym Netball  Row Swim T400m Tennis TSprnt WPolo
## 1 160  50  0.00  0.01 0.07    0.20 0.03 0.04  0.35   0.19   0.10  0.00
## 2 160  75  0.00  0.72 0.00    0.07 0.05 0.03  0.00   0.01   0.11  0.01
## 3 160 100  0.00  1.00 0.00    0.00 0.00 0.00  0.00   0.00   0.00  0.00
## 4 180  50  0.09  0.00 0.00    0.02 0.01 0.01  0.77   0.08   0.01  0.00
## 5 180  75  0.08  0.05 0.00    0.15 0.26 0.16  0.09   0.04   0.10  0.07
## 6 180 100  0.00  0.85 0.00    0.01 0.06 0.02  0.00   0.00   0.02  0.05
## 7 200  50  0.69  0.00 0.00    0.00 0.00 0.00  0.30   0.01   0.00  0.00
## 8 200  75  0.89  0.00 0.00    0.00 0.02 0.01  0.05   0.01   0.00  0.01
## 9 200 100  0.27  0.04 0.00    0.01 0.21 0.06  0.00   0.00   0.01  0.40
```

Better. Much better.

(g) For an athlete who is 180 cm tall and weighs 100 kg, what sport would you guess they play? How sure are you that you are right? Explain briefly.

Solution

Find this height and weight in your predictions (it's row 6). Look along the line for the highest probability, which is 0.85 for `Field` (that is, field athletics). All the other probabilities are much smaller (the biggest of the others is 0.06). So this means we would guess the athlete to be a field athlete, and because the predicted probability is so big, we are very likely to be right. This kind of thought process is characteristic of discriminant analysis, which we'll see more of later in the course. Compare that with the scatterplot you drew earlier: the field athletes do seem to be distinct from the rest in terms of height-weight combination. Some of the other height-weight combinations are almost equally obvious: for example, very tall people who are not very heavy are likely to play basketball. 400m runners are likely to be of moderate height but light weight. Some of the other sports, or height-weight combinations, are difficult to judge. Consider also that we have mixed up males and females in this data set. We might gain some clarity by including `Sex` in the model, and also in the predictions. But I wanted to keep this question relatively simple for you, and I wanted to stop from getting unreasonably long. (You can decide whether you think it's already too long.)

18
Survival analysis

```
library(survival)
library(tidyverse)
```

```
## -- Attaching packages ---- tidyverse 1.2.1 --
```

```
## v ggplot2 3.1.0     v purrr   0.2.5
## v tibble  1.4.2     v dplyr   0.7.8
## v tidyr   0.8.1     v stringr 1.3.1
## v readr   1.1.1     v forcats 0.3.0
```

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

```
library(survminer)
```

```
## Loading required package: ggpubr
```

```
## Loading required package: magrittr
```

```
##
## Attaching package: 'magrittr'
```

```
## The following object is masked from 'package:purrr':
##
##     set_names
```

```
## The following object is masked from 'package:tidyr':
##
##     extract
```

18.1 The Worcester survey

The Worcester survey was a long-term study of all myocardial-infarction victims admitted to hospitals in the Worcester, Massachusetts area. The data have been well studied, and can be found in the file link.

Heart attack.

Worcester is pronounced, by locals, *Woo-stuh*.

(a) Read the data and display the first few rows of the data frame. You might get an extra column, which you can ignore. For your information, the variables are:

- patient ID code
- admission date
- date of last followup (this is the date of death if the patient died)
- length of hospital stay (days)
- followup time (days) (time between admission and last followup)
- followup status: 1=dead, 0=alive
- Age in years (at admission)
- gender (0=male, 1=female)
- body mass index (kg/m^2)

Solution

```
my_url = "http://www.utsc.utoronto.ca/~butler/d29/whas100.csv"
whas100 = read_csv(my_url)
```

```
## Warning: Missing column names filled in:
## 'X1' [1]
```

```
## Parsed with column specification:
## cols(
##   X1 = col_integer(),
##   id = col_integer(),
##   admitdate = col_character(),
##   foldate = col_character(),
##   los = col_integer(),
##   lenfol = col_integer(),
##   fstat = col_integer(),
##   age = col_integer(),
##   gender = col_integer(),
##   bmi = col_double()
## )
```

```
whas100
```

```
## # A tibble: 100 x 10
##       X1    id admitdate foldate   los lenfol
##    <int> <int> <chr>     <chr>   <int>  <int>
##  1     1     1 3/13/1995 3/19/1~     4      6
```

```
## 2     2     2 1/14/1995 1/23/1~     5    374
## 3     3     3 2/17/1995 10/4/2~     5   2421
## 4     4     4 4/7/1995  7/14/1~     9     98
## 5     5     5 2/9/1995  5/29/1~     4   1205
## 6     6     6 1/16/1995 9/11/2~     7   2065
## 7     7     7 1/17/1995 10/15/~     3   1002
## 8     8     8 11/15/19~ 11/24/~    56   2201
## 9     9     9 8/18/1995 2/23/1~     5    189
## 10    10    10 7/22/1995 12/31/~     9   2719
## # ... with 90 more rows, and 4 more
## #   variables: fstat <int>, age <int>,
## #   gender <int>, bmi <dbl>
```

I seem to have an extra column called `X1`. This is because I saved my version of the data using the old `write.csv`, which comes with row names, and I forgot to get rid of them. These came back as an extra unnamed variable to which `read_delim` gave the name `X1`.

(b) Create a suitable response variable for a Cox proportional hazards model for time of survival, using the followup time and followup status.

Solution

`Surv`. The event here is death, so the two parts of the response variable are followup time `lenfol` and followup status, 1 being "dead", `fstat`:

```
y = with(whas100, Surv(lenfol, fstat == 1))
y
```

```
##   [1]    6   374  2421    98  1205  2065
##   [7] 1002  2201   189  2719+ 2638+  492
##  [13]  302  2574+ 2610+ 2641+ 1669  2624
##  [19] 2578+ 2595+  123  2613+  774  2012
##  [25] 2573+ 1874  2631+ 1907   538   104
##  [31]    6  1401  2710   841   148  2137+
##  [37] 2190+ 2173+  461  2114+ 2157+ 2054+
##  [43] 2124+ 2137+ 2031  2003+ 2074+  274
##  [49] 1984+ 1993+ 1939+ 1172    89   128
##  [55] 1939+   14  1011  1497  1929+ 2084+
##  [61]  107   451  2183+ 1876+  936   363
##  [67] 1048  1889+ 2072+ 1879+ 1870+ 1859+
##  [73] 2052+ 1846+ 2061+ 1912+ 1836+  114
##  [79] 1557  1278  1836+ 1916+ 1934+ 1923+
##  [85]   44  1922+  274  1860+ 1806  2145+
##  [91]  182  2013+ 2174+ 1624   187  1883+
##  [97] 1577    62  1969+ 1054
```

Just using `fstat` alone as the second thing in `Surv` also works, because anything that gives `TRUE` or 1 when the event (death) occurs is equally good. (In R, `TRUE` as a number is 1 and `FALSE` as a number is 0.)

I listed the values by way of checking. The ones with a + are censored: that is, the patient was still alive the last time the doctor saw them. Most of the censored values are longer times. Usually this happens because the patient was still alive at the end of the study.

(c) Fit a Cox proportional hazards model predicting survival time from age, gender and BMI. Obtain the `summary` (but you don't need to comment on it yet).

Solution

This, using the response variable that we just created:

```
whas100.1 = coxph(y ~ age + gender + bmi, data = whas100)
summary(whas100.1)
```

```
## Call:
## coxph(formula = y ~ age + gender + bmi, data = whas100)
##
##   n= 100, number of events= 51
##
##            coef exp(coef) se(coef)      z
## age     0.03713   1.03783  0.01272  2.918
## gender  0.14325   1.15402  0.30604  0.468
## bmi    -0.07083   0.93162  0.03607 -1.964
##        Pr(>|z|)
## age     0.00352 **
## gender  0.63973
## bmi     0.04956 *
## ---
## Signif. codes:
##   0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1  ' ' 1
##
##        exp(coef) exp(-coef) lower .95
## age       1.0378     0.9636    1.0123
## gender    1.1540     0.8665    0.6334
## bmi       0.9316     1.0734    0.8680
##        upper .95
## age       1.0640
## gender    2.1024
## bmi       0.9999
##
## Concordance= 0.683  (se = 0.037 )
```

```
## Rsquare= 0.194   (max possible= 0.985 )
## Likelihood ratio test= 21.54  on 3 df,   p=8e-05
## Wald test            = 19.46  on 3 df,   p=2e-04
## Score (logrank) test = 20.82  on 3 df,   p=1e-04
```

(d) Test the overall fit of the model. What does the result mean?

Solution

Look at those three P-values at the bottom. They are all small, so something in the model is helping to predict survival. As to what? Well, that's the next part.

(e) Can any of your explanatory variables be removed from the model? Explain briefly.

Solution

`gender` has a (very) large P-value, so that can be taken out of the model. The other two variables have small P-values (`bmi` only just under 0.05), so they need to stay. The other way to think about this is `step`, or `drop1`:

```
drop1(whas100.1, test = "Chisq")
```

```
## Single term deletions
##
## Model:
## y ~ age + gender + bmi
##        Df    AIC    LRT Pr(>Chi)
## <none>    402.66
## age     1 410.25 9.5972 0.001949 **
## gender  1 400.87 0.2185 0.640218
## bmi     1 404.60 3.9389 0.047182 *
## ---
## Signif. codes:
##   0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1  ' ' 1
```

This is here the same as the output from `summary`, but where it scores is if you have a categorical explanatory variable like "treatment" with more than two levels: `drop1` will tell you about keeping or dropping it as a whole. If you prefer:

Our categorical variable *gender* has only two levels.

```
whas100.step = step(whas100.1, trace = 0)
summary(whas100.step)
```

```
## Call:
## coxph(formula = y ~ age + bmi, data = whas100)
##
```

```
##   n= 100, number of events= 51
##
##         coef exp(coef) se(coef)      z
## age  0.03927   1.04005  0.01187  3.309
## bmi -0.07116   0.93131  0.03614 -1.969
##     Pr(>|z|)
## age 0.000938 ***
## bmi 0.048952 *
## ---
## Signif. codes:
##   0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1  ' ' 1
##
##     exp(coef) exp(-coef) lower .95 upper .95
## age    1.0401     0.9615    1.0161    1.0645
## bmi    0.9313     1.0738    0.8676    0.9997
##
## Concordance= 0.681  (se = 0.037 )
## Rsquare= 0.192   (max possible= 0.985 )
## Likelihood ratio test= 21.32  on 2 df,   p=2e-05
## Wald test            = 19  on 2 df,   p=7e-05
## Score (logrank) test = 19.99  on 2 df,   p=5e-05
```

`gender` comes out, but the others stay. As usual, put `trace=1` or `trace=2` to get more output, which will look like a sequence of `drop1`'s one after the other.

(f) Remove your most non-significant explanatory variable from the model and fit again. Take a look at the results. Are all your remaining explanatory variables significant? (If all your explanatory variables were previously significant, you can skip this part.)

Solution

So, take out `gender`:

```
whas100.2 = update(whas100.1, . ~ . - gender)
summary(whas100.2)
```

```
## Call:
## coxph(formula = y ~ age + bmi, data = whas100)
##
##   n= 100, number of events= 51
##
##         coef exp(coef) se(coef)      z
## age  0.03927   1.04005  0.01187  3.309
## bmi -0.07116   0.93131  0.03614 -1.969
##     Pr(>|z|)
```

```
## age 0.000938 ***
## bmi 0.048952 *
## ---
## Signif. codes:
##   0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1  ' ' 1
##
##     exp(coef) exp(-coef) lower .95 upper .95
## age    1.0401     0.9615    1.0161    1.0645
## bmi    0.9313     1.0738    0.8676    0.9997
##
## Concordance= 0.681  (se = 0.037 )
## Rsquare= 0.192   (max possible= 0.985 )
## Likelihood ratio test= 21.32  on 2 df,   p=2e-05
## Wald test            = 19  on 2 df,   p=7e-05
## Score (logrank) test = 19.99  on 2 df,   p=5e-05
```

Both explanatory variables are significant: `age` definitely, `bmi` only just. This is the same model as `step` gave me.

(g) Calculate the 1st quartile, median, and 3rd quartiles of age and BMI. (`quantile`.) Round these off to the nearest whole number. (Do the rounding off yourself, though R has a function `round` that does this, which you can investigate if you want.) As an alternative, you can get these by passing the whole data frame into `summary`.

Solution

```r
quantile(whas100$age)
```

```
##    0%   25%   50%   75%  100%
## 32.00 59.75 71.00 80.25 92.00
```

```r
quantile(whas100$bmi)
```

```
##       0%      25%      50%      75%     100%
## 14.91878 23.53717 27.19158 30.34770 39.93835
```

or

```r
with(whas100, {
    print(quantile(age))
    print(quantile(bmi))
})
```

```
##    0%   25%   50%   75%  100%
## 32.00 59.75 71.00 80.25 92.00
##       0%      25%      50%      75%     100%
## 14.91878 23.53717 27.19158 30.34770 39.93835
```

(since there are two things in the `with`, you need the curly brackets, and since there are curly brackets you need to explicitly `print` the things you want to display.) Or, tidyverse-style:

```
whas100 %>% select(age, bmi) %>% summarize(ageq = list(quantile(age)),
    bmiq = list(quantile(bmi)))
```

```
## # A tibble: 1 x 2
##   ageq      bmiq
##   <list>    <list>
## 1 <dbl [5]> <dbl [5]>
```

These are list-columns containing the five quantiles for each variable, which we then have to pull apart:

```
whas100 %>% select(age, bmi) %>% summarize(ageq = list(quantile(age)),
    bmiq = list(quantile(bmi))) %>% unnest()
```

```
## # A tibble: 5 x 2
##    ageq  bmiq
##   <dbl> <dbl>
## 1  32    14.9
## 2  59.8  23.5
## 3  71    27.2
## 4  80.2  30.3
## 5  92    39.9
```

The reason for the `list` and the list-columns and the `unnest` is that `quantile` returns the five numbers of the five-number summary, rather than just one number (in which case it would be a regular `summarize`). This might strike you as a bit repetitive. There is also a `summarize_all`, which summarizes all the columns (after you've done the `select`). This requires a function to summarize with, which we need to write first:

```
lq = function(x) {
    list(quantile(x))
}
whas100 %>% select(age, bmi) %>% summarize_all(lq) %>%
    unnest()
```

```
## # A tibble: 5 x 2
##     age   bmi
##   <dbl> <dbl>
## 1  32    14.9
## 2  59.8  23.5
## 3  71    27.2
```

```
## 4  80.2  30.3
## 5  92    39.9
```

The `summary` approach is simply this (which I think I now like best of all):

```r
summary(whas100)
```

```
##        X1               id
##  Min.   :  1.00   Min.   :  1.00
##  1st Qu.: 25.75   1st Qu.: 25.75
##  Median : 50.50   Median : 50.50
##  Mean   : 50.50   Mean   : 50.50
##  3rd Qu.: 75.25   3rd Qu.: 75.25
##  Max.   :100.00   Max.   :100.00
##   admitdate           foldate
##  Length:100         Length:100
##  Class :character   Class :character
##  Mode  :character   Mode  :character
##
##
##
##       los            lenfol
##  Min.   : 1.00   Min.   :   6
##  1st Qu.: 4.00   1st Qu.: 715
##  Median : 5.00   Median :1878
##  Mean   : 6.84   Mean   :1505
##  3rd Qu.: 7.00   3rd Qu.:2076
##  Max.   :56.00   Max.   :2719
##      fstat           age
##  Min.   :0.00   Min.   :32.00
##  1st Qu.:0.00   1st Qu.:59.75
##  Median :1.00   Median :71.00
##  Mean   :0.51   Mean   :68.25
##  3rd Qu.:1.00   3rd Qu.:80.25
##  Max.   :1.00   Max.   :92.00
##      gender          bmi
##  Min.   :0.00   Min.   :14.92
##  1st Qu.:0.00   1st Qu.:23.54
##  Median :0.00   Median :27.19
##  Mean   :0.35   Mean   :27.04
##  3rd Qu.:1.00   3rd Qu.:30.35
##  Max.   :1.00   Max.   :39.94
```

or, if you want just the summaries of `age` and `bmi`, you select those columns first before passing the data frame into `summary`:

```
whas100 %>% select(age, bmi) %>% summary()
```

```
##       age             bmi
##  Min.   :32.00   Min.   :14.92
##  1st Qu.:59.75   1st Qu.:23.54
##  Median :71.00   Median :27.19
##  Mean   :68.25   Mean   :27.04
##  3rd Qu.:80.25   3rd Qu.:30.35
##  Max.   :92.00   Max.   :39.94
```

Think small tools, combined.

Whichever of this multitude of ways appeals to you:

60, 71 and 80 for age, 24, 27 and 30 for BMI.

(h) Make a data frame out of all the combinations of age and BMI values (that you obtained in the previous part) suitable for predicting with.

Solution

The inevitable `crossing`:

```
whas100.new = crossing(age = c(60, 71, 80), bmi = c(24,
    27, 30))
whas100.new
```

```
## # A tibble: 9 x 2
##     age   bmi
##   <dbl> <dbl>
## 1    60    24
## 2    60    27
## 3    60    30
## 4    71    24
## 5    71    27
## 6    71    30
## 7    80    24
## 8    80    27
## 9    80    30
```

Or, with some setup beforehand to make the `crossing` clearer. This is my no-think approach:

```
ages = c(60, 71, 80)
bmis = c(24, 27, 30)
whas100.new = crossing(age = ages, bmi = bmis)
whas100.new
```

```
## # A tibble: 9 x 2
```

```
##     age   bmi
##   <dbl> <dbl>
## 1    60    24
## 2    60    27
## 3    60    30
## 4    71    24
## 5    71    27
## 6    71    30
## 7    80    24
## 8    80    27
## 9    80    30
```

(i) Obtain predicted survival probabilities for each of the values in your new data frame. Use your best model. (You don't need to look at the results, though you can if you want to.)

Solution

The magic word is `survfit` (which plays the role of `predict` here). The best model is `whas100.2`, with the non-significant `gender` removed:

```
pp2 = survfit(whas100.2, whas100.new, data = whas100)
```

This doesn't need the `data=` at the end (it works perfectly well without), but the plot (later) seems to need it to be there. I think the plot needs the information from the original data to be in the predictions somewhere.

This is kind of long to look at (`summary(pp2)` would be the thing), so we will need to make a graph of it. I gave it a name, since I want to use it again later.

(j) Make a graph depicting the survival curves from `survfit` with different colours distinguishing the different survival curves.

Solution

This is actually easy once you work out what to do:

```
ggsurvplot(pp2, conf.int = F)
```

Without the `conf.int` thing, you get confidence intervals for each survival curve, which overlap each other and generally make the plot look messy.

The "strata" are the different age-BMI combinations that you predicted for, so it's usually a good idea to list the "new" prediction data frame, either here or when you assess the effects of the variables (next part) so that you can see which is which:

```
whas100.new
```

```
## # A tibble: 9 x 2
##     age   bmi
##   <dbl> <dbl>
## 1    60    24
## 2    60    27
## 3    60    30
## 4    71    24
## 5    71    27
## 6    71    30
## 7    80    24
## 8    80    27
## 9    80    30
```

(k) What is the effect of age on survival? What is the effect of BMI on survival? Explain briefly. (You will have to disentangle the meaning of the different coloured lines on the plot to do this.)

Solution

Bear in mind that up-and-to-the-right is best for a survival curve, since that means that people in the upper-right group have a higher chance of surviving for longer.

The best survival curve is therefore the olive-green one. According to the legend, this goes with stratum 3, which is (according to

the listing of `whas100.new`) age 60 (the youngest) and BMI 30 (the highest). So it looks as if the best survival goes with a lower age (not surprising) and a higher BMI (surprising; see discussion about BMI below).

You can also leave one variable constant and see the effects of changing the other one. Let's pick the oldest age 80: the BMI values are 24 (stratum 7, blue), 27 (stratum 8, purple), 30 (stratum 9, pink). These survival curves are the bottom one, the second bottom one, and the fourth bottom one. At this age, survival chances are not great, but having a higher BMI goes with a greater chance of surviving longer.

Or pick a BMI, say 30. These are strata 3 (olive green), 6 (light blue) and 9 (pink) respectively for ages 60, 71 and 80. These are the best, 3rd best and 5th best survival curves; that is, as age increases, the chance of surviving a long time decreases.

The effect of BMI, though, seems backwards: a higher BMI is associated with a *higher* chance of survival.

That's the end of what I wanted you to do, but:

A higher BMI is usually associated with being obese (and therefore unhealthy), so you'd expect the effect of BMI to be the other way around. According to Wikipedia (link), the BMI values here are "overweight" or close to it. Maybe being heavier helps the body recover from a heart attack.

Let's start with the martingale residual plot:

```
ggcoxdiagnostics(whas100.2) + geom_smooth()
```

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

There is a suspicion of bendiness here, though the left side of the curve is entirely because of that one positive residual on the left. In any case, this suggests that nonlinearity (evidently in terms of BMI,

since that's the relationship that currently makes no sense) would be worth exploring.

Thus:

```
whas100.3 = update(whas100.2, . ~ . + I(bmi^2))
summary(whas100.3)
```

```
## Call:
## coxph(formula = y ~ age + bmi + I(bmi^2), data = whas100)
##
##   n= 100, number of events= 51
##
##               coef exp(coef)  se(coef)
## age       0.040542  1.041375  0.012035
## bmi      -0.848949  0.427864  0.231562
## I(bmi^2)  0.014500  1.014606  0.004227
##               z Pr(>|z|)
## age       3.369 0.000755 ***
## bmi      -3.666 0.000246 ***
## I(bmi^2)  3.430 0.000603 ***
## ---
## Signif. codes:
##   0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1  ' ' 1
##
##          exp(coef) exp(-coef) lower .95
## age         1.0414     0.9603    1.0171
## bmi         0.4279     2.3372    0.2718
## I(bmi^2)    1.0146     0.9856    1.0062
##          upper .95
## age         1.0662
## bmi         0.6736
## I(bmi^2)    1.0230
##
## Concordance= 0.693  (se = 0.04 )
## Rsquare= 0.264   (max possible= 0.985 )
## Likelihood ratio test= 30.71  on 3 df,   p=1e-06
## Wald test            = 32.56  on 3 df,   p=4e-07
## Score (logrank) test = 36.57  on 3 df,   p=6e-08
```

Ah, that seems to be it. The significant positive coefficient on `bmi`-squared means that the "hazard of dying" increases faster with increasing `bmi`, so there ought to be an optimal BMI beyond which survival chances decrease again. Let's explore that on a graph.

I'm going to focus on a close-to-median age of 70, since, in this model, the effect of BMI is the same for all ages (to make it different, we would need an interaction term, ANOVA-style).

First we create a data frame with a bunch of different BMIs in, and one age 70:

```
bmis = seq(20, 36, 4)
ages = 70
whas100.new.2 = crossing(bmi = bmis, age = ages)
whas100.new.2
```

```
## # A tibble: 5 x 2
##     bmi   age
##   <dbl> <dbl>
## 1    20    70
## 2    24    70
## 3    28    70
## 4    32    70
## 5    36    70
```

It is rather absurd to have a plural `ages` with only one age in it, but that's the way it goes, if you're me and trying to avoid thinking.

Predictions, using the model with the squared term in it:

```
pp3 = survfit(whas100.3, whas100.new.2, data = whas100)
```

And then the plot:

```
ggsurvplot(pp3, conf.int = F)
```

and the customary reminder of which stratum is which, with its rather ungainly name:

```
whas100.new.2
```

```
## # A tibble: 5 x 2
##     bmi   age
```

```
##   <dbl> <dbl>
## 1    20    70
## 2    24    70
## 3    28    70
## 4    32    70
## 5    36    70
```

This time, the green survival curve is best, stratum 3, which means that survival is best at BMI 28, and worse for both higher BMIs and lower BMIs. You can follow the sequence of colours: red, olive-green, green, blue, pink, that goes up and then down again. But it's still true that having a very *low* BMI is worst, which is why our (linear) model said that having a higher BMI was better.

It would have been better to have you put a squared term in the model, but the question was already long and complicated enough, and I didn't want to make your lives more of a nightmare than they are already becoming!

18.2 *Drug treatment programs*

One of the goals of drug treatment programs is to lengthen the time until the patient returns to using drugs. (It is not generally possible to prevent patients from *ever* using drugs again.) In one study, over 600 former drug users took part. Two different programs, a short program and a long program, were offered at two different sites, labelled A and B. The data can be found in link. The variables are these:

- `ID`: patient ID number
- `age`: patient age at enrollment into the study
- `ndrugtx`: number of previous drug treatments
- `treat`: 0 for short treatment program, 1 for long program
- `site`: 0 for site A, 1 for site B
- `time`: time until return to drug use
- `censor`: whether the subject returned to drug use (1) or not (0) during the follow-up period
- `herco`: whether subject used heroine or cocaine in the last 3 months: 1 is both, 2 is one (either heroine or cocaine), 3 is neither.

(a) Read in the data and check in one way or another that you have what was promised above.

Solution

This:

```
my_url = "http://www.utsc.utoronto.ca/~butler/d29/drugusers.txt"
drugusers = read_delim(my_url, " ")

## Parsed with column specification:
## cols(
##   row = col_integer(),
##   ID = col_integer(),
##   age = col_integer(),
##   ndrugtx = col_integer(),
##   treat = col_integer(),
##   site = col_integer(),
##   time = col_integer(),
##   censor = col_integer(),
##   herco = col_integer()
## )

drugusers

## # A tibble: 628 x 9
##      row    ID   age ndrugtx treat  site
##    <int> <int> <int>   <int> <int> <int>
##  1     1     1    39       1     1     0
##  2     2     2    33       8     1     0
##  3     3     3    33       3     1     0
##  4     4     4    32       1     0     0
##  5     5     5    24       5     1     0
##  6     6     6    30       1     1     0
##  7     7     7    39      34     1     0
##  8     8     8    27       2     1     0
##  9     9     9    40       3     1     0
## 10    10    10    36       7     1     0
## # ... with 618 more rows, and 3 more
## #   variables: time <int>, censor <int>,
## #   herco <int>
```

This shows that you have over 600 rows and the variables described.

(b) There are some missing values in the dataframe. Demonstrate this using `summary`. Pipe the dataframe into `drop_na` and show that they have gone. (`drop_na` removes all rows that have missing values in them.)

Solution

First off, `summary` is a quick way to show how many missing values there are:

```
summary(drugusers)
```

```
##       row              ID
##  Min.   :  1.0   Min.   :  1.0
##  1st Qu.:157.8   1st Qu.:157.8
##  Median :314.5   Median :314.5
##  Mean   :314.5   Mean   :314.5
##  3rd Qu.:471.2   3rd Qu.:471.2
##  Max.   :628.0   Max.   :628.0
##
##       age           ndrugtx
##  Min.   :20.00   Min.   : 0.000
##  1st Qu.:27.00   1st Qu.: 1.000
##  Median :32.00   Median : 3.000
##  Mean   :32.37   Mean   : 4.574
##  3rd Qu.:37.00   3rd Qu.: 6.000
##  Max.   :56.00   Max.   :40.000
##  NA's   :5       NA's   :17
##      treat             site
##  Min.   :0.0000   Min.   :0.000
##  1st Qu.:0.0000   1st Qu.:0.000
##  Median :0.0000   Median :0.000
##  Mean   :0.4904   Mean   :0.293
##  3rd Qu.:1.0000   3rd Qu.:1.000
##  Max.   :1.0000   Max.   :1.000
##
##       time            censor
##  Min.   :   2.0   Min.   :0.0000
##  1st Qu.:  79.0   1st Qu.:1.0000
##  Median : 166.0   Median :1.0000
##  Mean   : 234.7   Mean   :0.8089
##  3rd Qu.: 365.2   3rd Qu.:1.0000
##  Max.   :1172.0   Max.   :1.0000
##
##      herco
##  Min.   :1.000
##  1st Qu.:1.000
##  Median :2.000
##  Mean   :1.898
##  3rd Qu.:3.000
##  Max.   :3.000
```

```
##
```

Age has five missing values and "number of previous drug treatments" has seventeen.

Following the instructions, and saving back into the original dataframe:

```
drugusers <- drugusers %>% drop_na()
```

and then

```
summary(drugusers)
```

```
##       row              ID
##  Min.   :  1.0   Min.   :  1.0
##  1st Qu.:155.2   1st Qu.:155.2
##  Median :312.5   Median :312.5
##  Mean   :313.8   Mean   :313.8
##  3rd Qu.:473.8   3rd Qu.:473.8
##  Max.   :628.0   Max.   :628.0
##       age           ndrugtx
##  Min.   :20.00   Min.   : 0.000
##  1st Qu.:27.00   1st Qu.: 1.000
##  Median :32.00   Median : 3.000
##  Mean   :32.39   Mean   : 4.579
##  3rd Qu.:37.00   3rd Qu.: 6.000
##  Max.   :56.00   Max.   :40.000
##      treat             site
##  Min.   :0.0000   Min.   :0.0000
##  1st Qu.:0.0000   1st Qu.:0.0000
##  Median :0.0000   Median :0.0000
##  Mean   :0.4918   Mean   :0.2984
##  3rd Qu.:1.0000   3rd Qu.:1.0000
##  Max.   :1.0000   Max.   :1.0000
##       time            censor
##  Min.   :   2.0   Min.   :0.0000
##  1st Qu.:  79.0   1st Qu.:1.0000
##  Median : 166.0   Median :1.0000
##  Mean   : 234.4   Mean   :0.8115
##  3rd Qu.: 361.8   3rd Qu.:1.0000
##  Max.   :1172.0   Max.   :1.0000
##      herco
##  Min.   :1.00
##  1st Qu.:1.00
##  Median :2.00
##  Mean   :1.89
```

```
##  3rd Qu.:3.00
##  Max.   :3.00
```

No NA left. Gosh, as they say, that was easy.

Extra: how many rows did we lose:

```
nrow(drugusers)
```

```
## [1] 610
```

There were 628 rows before, so we lost 18. (There were 22 missing values, but some of them were two on one row, so we only lost 18 rows.)

This is a very unsophisticated way of dealing with missing values. Another way is to "impute" them, that is, to guess what they would have been, and then fill in the guessed values and use them as if they were the truth.

(c) Some of these variables are recorded as numbers but are actually categorical. Which ones? Re-define these variables in your data frame so that they have sensible (text) values.

Solution

These variables are actually categorical rather than quantitative:

- `treat`
- `site`
- `censor`
- `herco`

Most of them have only two levels, so it doesn't matter whether we make them categorical or leave them as numbers, but for `herco` it matters. Let's give them all sensible values, mostly with `ifelse`, thus:

```
drugusers <- drugusers %>% mutate(treat = ifelse(treat ==
    0, "short", "long"), site = ifelse(site ==
    0, "A", "B"), censor = ifelse(censor == 1,
    "returned", "no-return"), herco = case_when(herco ==
    1 ~ "both", herco == 2 ~ "one", herco == 3 ~
    "neither"))
```

I'm living on the edge and overwriting everything:

```
drugusers
```

```
## # A tibble: 610 x 9
##      row    ID   age ndrugtx treat site
##    <int> <int> <int>   <int> <chr> <chr>
##  1     1     1    39       1 long  A
##  2     2     2    33       8 long  A
##  3     3     3    33       3 long  A
##  4     4     4    32       1 short A
##  5     5     5    24       5 long  A
##  6     6     6    30       1 long  A
##  7     7     7    39      34 long  A
##  8     8     8    27       2 long  A
##  9     9     9    40       3 long  A
## 10    10    10    36       7 long  A
## # ... with 600 more rows, and 3 more
## #   variables: time <int>, censor <chr>,
## #   herco <chr>
```

(d) Create a suitable reponse variable for a Cox proportional hazards regression that predicts time until return to drug use from the other variables. This requires some care, because you need to be sure about what the censoring variable actually represents and what you need it to represent.

Solution

This is `Surv` in package `survival`. The response variable needs to encode two things: the time until the event of interest (return to drug use) and whether or not that event happened for each patient. In this case, that is `censor="returned"`.

```
y = with(drugusers, Surv(time, censor == "returned"))
```

Use whatever name you gave to the level of `censor` that means "returned to drug use".

(e) Look at the first few values of your response variable. Why is the fifth one marked with a +? Explain briefly.

Solution

`head` works as well with a vector (displaying the first six values) as it does with a data frame:

```
head(y)
```

```
## [1] 188   26  207  144  551+  32
```

The fifth value is marked with a + because it is a censored value: this is a patient who was never observed to go back to drug use. You can tell this by looking at the `head` of the entire data frame:

```
head(drugusers)
```

```
## # A tibble: 6 x 9
##     row    ID   age ndrugtx treat site   time
##   <int> <int> <int>   <int> <chr> <chr> <int>
## 1     1     1    39       1 long  A       188
## 2     2     2    33       8 long  A        26
## 3     3     3    33       3 long  A       207
## 4     4     4    32       1 short A       144
## 5     5     5    24       5 long  A       551
## 6     6     6    30       1 long  A        32
## # ... with 2 more variables: censor <chr>,
## #   herco <chr>
```

since this patient has `censor="no-return"`. The other ones have `censor="returned"`; these are all "uncensored" in the jargon.

Typically, censored values will be bigger than uncensored ones, because (in general) the individual will be observed until the study ends, and studies of this kind carry on for years:

```
ggplot(drugusers, aes(x = censor, y = time)) +
    geom_boxplot()
```

Yep. The smallest time for a censored observation would be an upper outlier if it were observed for an uncensored observation.

One nice side-effect of turning `censor` into a categorical variable is that it can now distinguish groups as a boxplot requires.

I discovered something rather amusing when I originally wrote this (a year ago). Suppose you want to compare times for the two treatment groups, and you *also* want to distinguish censored from non-censored observations. Then, this works:

```
ggplot(drugusers, aes(x = treat, y = time, colour = censor)) +
    geom_boxplot()
```

For each treatment, you get side-by-side boxplots of the times for censored (red) and uncensored (blue) observations, and so you see for both treatments (short and long) the censored times are typically longer than the uncensored ones.

(This you may recognize as a "grouped boxplot", for when we have two categorical variables and one quantitative one.)

I borrow this idea for two-way ANOVA (coming up later).

(f) Fit a Cox proportional hazards model, predicting from all the other variables (except for `row` and `ID`) that you haven't used yet. Display the results.

Solution

```
drugusers.1 = coxph(y ~ age + ndrugtx + treat +
    site + herco, data = drugusers)
summary(drugusers.1)
```

```
## Call:
## coxph(formula = y ~ age + ndrugtx + treat + site + herco, data = drugusers)
##
##   n= 610, number of events= 495
##
##                    coef exp(coef)  se(coef)
## age           -0.023798  0.976483  0.007561
## ndrugtx        0.034815  1.035429  0.007755
## treatshort     0.254606  1.289953  0.091006
## siteB         -0.173021  0.841120  0.102105
## herconeither   0.125779  1.134032  0.103075
```

```
## hercoone      0.247318  1.280586  0.122759
##                    z Pr(>|z|)
## age          -3.148  0.00165 **
## ndrugtx       4.490 7.14e-06 ***
## treatshort    2.798  0.00515 **
## siteB        -1.695  0.09016 .
## herconeither  1.220  0.22236
## hercoone      2.015  0.04394 *
## ---
## Signif. codes:
##   0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1  ' ' 1
##
##              exp(coef) exp(-coef) lower .95
## age             0.9765     1.0241    0.9621
## ndrugtx         1.0354     0.9658    1.0198
## treatshort      1.2900     0.7752    1.0792
## siteB           0.8411     1.1889    0.6886
## herconeither    1.1340     0.8818    0.9266
## hercoone        1.2806     0.7809    1.0067
##              upper .95
## age             0.9911
## ndrugtx         1.0513
## treatshort      1.5418
## siteB           1.0275
## herconeither    1.3879
## hercoone        1.6289
##
## Concordance= 0.581  (se = 0.014 )
## Rsquare= 0.056   (max possible= 1 )
## Likelihood ratio test= 35.08  on 6 df,   p=4e-06
## Wald test            = 36.96  on 6 df,   p=2e-06
## Score (logrank) test = 37.36  on 6 df,   p=1e-06
```

Another way to handle "all the other x's except `row`, `ID`, `time` and `censor`" is this:

```
drugusers.1a = coxph(y ~ . - row - ID - time -
    censor, data = drugusers)
broom::tidy(drugusers.1a)
```

```
## # A tibble: 6 x 7
##   term  estimate std.error statistic p.value
##   <chr>    <dbl>     <dbl>     <dbl>   <dbl>
## 1 age    -0.0238   0.00756     -3.15 1.65e-3
## 2 ndru~   0.0348   0.00775      4.49 7.14e-6
```

```
## 3 trea~   0.255    0.0910       2.80 5.15e-3
## 4 siteB  -0.173    0.102       -1.69 9.02e-2
## 5 herc~   0.126    0.103        1.22 2.22e-1
## 6 herc~   0.247    0.123        2.01 4.39e-2
## # ... with 2 more variables: conf.low <dbl>,
## #   conf.high <dbl>
```

Same. I used `tidy` from `broom` to shorten the output a bit.

(g) Find which explanatory variables can be removed at $\alpha = 0.05$ (there should be two of them). Bear in mind that we have categorical variables, so that looking at the output from `summary` is not enough.

Solution

The hint is meant to suggest to you that looking at `drop1` is the right way to go:

```r
drop1(drugusers.1, test = "Chisq")
```

```
## Single term deletions
##
## Model:
## y ~ age + ndrugtx + treat + site + herco
##         Df    AIC     LRT  Pr(>Chi)
## <none>     5712.2
## age      1 5720.3 10.0972  0.001485 **
## ndrugtx  1 5727.3 17.1446 3.464e-05 ***
## treat    1 5718.0  7.8352  0.005124 **
## site     1 5713.1  2.9335  0.086760 .
## herco    2 5712.5  4.3182  0.115427
## ---
## Signif. codes:
##   0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1  ' ' 1
```

Note that `herco`, a categorical variable with three levels, has 2 degrees of freedom here, since a test of "no effect of `herco`" is testing that survival is the same at *all three* levels of `herco`.

(h) Remove *all* the non-significant explanatory variables and re-fit your model. By carrying out a suitable test demonstrate that your smaller model is the better one.

Solution

`site` and `herco` are the two variables to come out. I like `update`, but there is no problem about copying-pasting your `coxph` and taking out what you no longer need.

The researchers were probably relieved that there was not quite a significant effect of *site*.

```
drugusers.2 = update(drugusers.1, . ~ . - site -
    herco)
```

Having fit two models, we can use `anova` to compare them. The right test gets done, so no need for `test=`:

```
anova(drugusers.2, drugusers.1)
```

```
## Analysis of Deviance Table
##  Cox model: response is  y
##  Model 1: ~ age + ndrugtx + treat
##  Model 2: ~ age + ndrugtx + treat + site + herco
##    loglik  Chisq Df P(>|Chi|)
## 1 -2853.7
## 2 -2850.1 7.2117  3   0.06545 .
## ---
## Signif. codes:
##   0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1  ' ' 1
```

There is no significant difference between these two models, so we can go with the smaller, simpler one (with just `age`, `ndrugtx` and `treat`).

Not at the 0.05 level, anyway.

(i)* Display your better model. Are all of the explanatory variables significant? Do their slope coefficients have sensible signs (plus or minus), based on what you know or can guess about drug treatments? Explain briefly.

Solution

```
summary(drugusers.2)
```

```
## Call:
## coxph(formula = y ~ age + ndrugtx + treat, data = drugusers)
##
##   n= 610, number of events= 495
##
##                 coef exp(coef)  se(coef)
## age        -0.020801  0.979414  0.007419
## ndrugtx     0.035567  1.036207  0.007621
## treatshort  0.231055  1.259929  0.090175
##                 z Pr(>|z|)
## age        -2.804  0.00505 **
## ndrugtx     4.667 3.05e-06 ***
## treatshort  2.562  0.01040 *
## ---
## Signif. codes:
##   0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1  ' ' 1
```

```
##
##            exp(coef) exp(-coef) lower .95
## age           0.9794     1.0210    0.9653
## ndrugtx       1.0362     0.9651    1.0208
## treatshort    1.2599     0.7937    1.0558
##            upper .95
## age           0.9938
## ndrugtx       1.0518
## treatshort    1.5035
##
## Concordance= 0.572  (se = 0.014 )
## Rsquare= 0.045   (max possible= 1 )
## Likelihood ratio test= 27.87  on 3 df,   p=4e-06
## Wald test            = 30.5  on 3 df,   p=1e-06
## Score (logrank) test = 30.62  on 3 df,   p=1e-06
```

The three remaining explanatory variables are all clearly significant: the patient's age, the number of previous drug treatments, and whether the treatment was short or long. This is legit (we don't need to run `drop1` again) because the remaining explanatory variables are all quantitative or have only two levels, so that the single-df tests in `summary` are what we need.

Do their slope coefficients have sensible signs? Well, this requires careful thought. A positive coefficient means that increasing that variable *increases the hazard of the event*: ie., it makes the event likelier to happen sooner. Here, the "event" is "return to drug use":

- Age has a negative coefficient, so an older person is likely to take *longer* to return to drug use, other things being equal. This makes some kind of sense, if you imagine drug use as being related to maturity, or an older drug user as being more strongly committed to "turning their life around", so that a drug treatment of any kind is going to be more effective on an older patient.

- The number of previous treatments has a positive coefficient, so that a patient who has had a lot of previous treatments is likely to go back to drugs sooner. Such a person might be an "addict" for whom treatments really do not work, or might not be committed to giving up drugs.

- `treatshort` has a positive coefficient. This says that if you give a patient a short treatment, they are more likely (other things being equal) to go back to drugs sooner, as compared to the baseline long treatment. That is, a longer treatment is more effective than a shorter one. Given a significant effect of treatment length, this is the way around you would expect it to be.

(j) We have three variables left in our model, `age`, `ndrugtx` and `treat`. The quartiles of age are 27 and 37, the quartiles of `ndrugtx` are 1 and 6, and the two possible values of `treat` are `short` and `long`. Create a data frame with variables of these names and all possible combinations of their values (so there should be 8 rows in the resulting data frame). Display the resulting data frame.

Solution

This data frame is going to be used for prediction, so I will call it `new` and construct it in pieces as I did before (thus meaning that I don't have to think too hard about what I'm doing):

```
ages = c(27, 37)
ndrugtxs = c(1, 6)
treats = c("short", "long")
new = crossing(age = ages, ndrugtx = ndrugtxs,
    treat = treats)
new
```

```
## # A tibble: 8 x 3
##     age ndrugtx treat
##   <dbl>   <dbl> <chr>
## 1    27       1 long
## 2    27       1 short
## 3    27       6 long
## 4    27       6 short
## 5    37       1 long
## 6    37       1 short
## 7    37       6 long
## 8    37       6 short
```

8 rows as promised.

(k) Obtain predicted survival probabilities for each of the values of `age`, `ndrugtx` and `treat` used in the previous part. You don't need to display it (we are going to plot it shortly).

Solution

`survfit` is the survival analysis version of `predict` and works the same way, so this is all you need:

```
pp = survfit(drugusers.2, new, data = drugusers)
```

Make sure that you use your best model, ie. the second one. The `data=` is needed for the plot below, not in itself for the prediction.

(l) Plot your predicted survival curves.

Solution

This:

```
ggsurvplot(pp, conf.int = F)
```

The only thing to remember is that you plot your *predictions*, not the model from which they came.

If your plot didn't come out, you may have to go back and re-do the `survfit` with the `data=` at the end.

For reference in a minute:

```
new
```

```
## # A tibble: 8 x 3
##     age ndrugtx treat
##   <dbl>   <dbl> <chr>
## 1    27       1 long
## 2    27       1 short
## 3    27       6 long
## 4    27       6 short
## 5    37       1 long
## 6    37       1 short
## 7    37       6 long
## 8    37       6 short
```

(m) Which of your combinations of values is predicted to take the longest to return to drug use? Which is predicted to take the shortest time? Explain briefly.

Solution

Remember that "up and to the right" is the best survival curve: that is, the people on this survival curve are predicted to take the longest to return to drug use. On my plot, this is the pale blue survival curve, stratum 5. Going back to my combinations data frame,

this is 37-year-olds with only one previous drug treatment and the longer drug treatment this time.

The worst is my green survival curve, stratum 4, which is the exact opposite of this: 27-year-olds, 6 previous drug treatments, shorter treatment this time.

"Returning to drug use" is like "death" in that you want it to be a long time before it happens, so "best" is top right on the plot of survival curves. In other curcumstances, you might want the event to happen *sooner*, in which case the lower-left survival curve would be the "best" one.

(n) Are your survival curve plot and your conclusions from part (here) consistent, or not? Explain briefly.

Solution

The survival curves say that being older, having fewer previous treatments and being on the long treatment are better in terms of taking longer to return to drug use. Our analysis of whether the slope coefficients in `drugusers.2` were positive or negative came to exactly the same conclusion. So the survival curves and part (here) are entirely consistent.

On my plot with the legend, you can assess the effects of the individual variables: for example, to assess the effect of age, find two combos that differ only in age, say strata 1 and 5, the red and light blue ones. Of these, the light blue survival curve is higher, so age 37 is better in terms of survival than age 27. This will work whichever such pair you pick: for example, strata 3 and 7, the olive green and purple curves, compare the same way.

More comparison for you to do: to assess the effect of number of previous treatments, compare eg. strata 1 and 3, red and olive green, and to assess the effect of treatment length, compare eg. strata 5 and 6, light blue and darker blue.

All this struggling to identify colours makes me think of link, in which the guy behind the webcomic XKCD did a survey where he showed people a whole bunch of different colours and asked the people to name the colours.

18.3 Multiple myeloma

Multiple myeloma is a kind of cancer. It forms in a plasma cell (which is a type of white blood cell). It causes cancer cells to accumulate in the bone marrow, where they crowd out healthy blood cells. Plasma cells make antibodies (to help fight infections), while the cancer cells don't: they produce abnormal proteins that can cause kidney problems. (This adapted from link.) The variables are:

- `time`: survival time from diagnosis (months)
- `vstatus`: 0=alive, 1=dead at end of study
- `logbun`: log of BUN test score (BUN test is a test of kidney function, not to be confused with cha siu bao).
- `hgb`: hemoglobin (at diagnosis).
- `platelet`: platelets: 1=normal, 0=abnormal (at diagnosis).
- `age` at diagnosis, in years
- `logwbc`: log of WBC (white blood cell count, at diagnosis)
- `frac`: fractures at diagnosis (0=absent, 1=present)
- `logpbm`: log of percent of plasma cells in bone marrow
- `protein`: proteinuria (protein in urine) at diagnosis. Most people have very little, so a larger than normal amount indicates illness of some kind.
- `scalc`: serum calcium at diagnosis.

Barbecued pork in a bun. A staple of Chinese dim sum and Chinese bakeries, such as Ding Dong bakery on Spadina.

The data, on 65 patients with multiple myeloma, are in link. Some of the variables are logs because they could take very large values.

There are a lot of parts here, but each part is supposed to be short.

(a) Read in the data and display (some of) the values. Confirm that you have the right number of observations and the right variables.

Solution

The usual:

```
my_url = "http://www.utsc.utoronto.ca/~butler/d29/myeloma.csv"
myeloma = read_csv(my_url)
```

```
## Parsed with column specification:
## cols(
##   time = col_double(),
##   vstatus = col_double(),
##   logbun = col_double(),
##   hgb = col_double(),
##   platelet = col_double(),
##   age = col_double(),
##   logwbc = col_double(),
##   frac = col_double(),
##   logpbm = col_double(),
##   protein = col_double(),
##   scalc = col_double()
## )
```

```
myeloma
```

```
## # A tibble: 65 x 11
##     time vstatus logbun   hgb platelet   age
##    <dbl>   <dbl>  <dbl> <dbl>    <dbl> <dbl>
##  1  1.25       1   2.22   9.4        1    67
##  2  1.25       1   1.94  12          1    38
##  3  2          1   1.52   9.8        1    81
##  4  2          1   1.75  11.3        0    75
##  5  2          1   1.30   5.1        0    57
##  6  3          1   1.54   6.7        1    46
##  7  5          1   2.24  10.1        1    50
##  8  5          1   1.68   6.5        1    74
##  9  6          1   1.36   9          1    77
## 10  6          1   2.11  10.2        0    70
## # ... with 55 more rows, and 5 more
## #   variables: logwbc <dbl>, frac <dbl>,
## #   logpbm <dbl>, protein <dbl>, scalc <dbl>
```

65 observations, and all the variables listed. If you want to go further (not necessary here), you can check that the variables `vstatus`, `platelet` and `frac` that should be zero and one actually *are* zero and one, at least for the values shown (they are), and the ages look like ages (they do).

The tidyverse also offers:

```
glimpse(myeloma)
```

```
## Observations: 65
## Variables: 11
## $ time     <dbl> 1.25, 1.25, 2.00, 2.00,...
## $ vstatus  <dbl> 1, 1, 1, 1, 1, 1, 1, 1,...
## $ logbun   <dbl> 2.2175, 1.9395, 1.5185,...
## $ hgb      <dbl> 9.4, 12.0, 9.8, 11.3, 5...
## $ platelet <dbl> 1, 1, 1, 0, 0, 1, 1, 1,...
## $ age      <dbl> 67, 38, 81, 75, 57, 46,...
## $ logwbc   <dbl> 3.6628, 3.9868, 3.8751,...
## $ frac     <dbl> 1, 1, 1, 1, 1, 0, 1, 0,...
## $ logpbm   <dbl> 1.9542, 1.9542, 2.0000,...
## $ protein  <dbl> 12, 20, 2, 0, 3, 12, 4,...
## $ scalc    <dbl> 10, 18, 15, 12, 9, 10, ...
```

which gives a bit more of a picture of the values. Or if you were serious about checking, you could do

Don't confuse this with *glance* from *broom*, which gives a one-line summary of a *model*, containing things like R-squared and a test for the overall model significance.

```
summary(myeloma)
```

```
##       time          vstatus
##  Min.   : 1.25   Min.   :0.0000
##  1st Qu.: 7.00   1st Qu.:0.0000
##  Median :15.00   Median :1.0000
##  Mean   :24.01   Mean   :0.7385
##  3rd Qu.:35.00   3rd Qu.:1.0000
##  Max.   :92.00   Max.   :1.0000
##      logbun            hgb
##  Min.   :0.7782   Min.   : 4.9
##  1st Qu.:1.1461   1st Qu.: 8.8
##  Median :1.3222   Median :10.2
##  Mean   :1.3929   Mean   :10.2
##  3rd Qu.:1.5682   3rd Qu.:12.0
##  Max.   :2.2355   Max.   :14.6
##     platelet           age
##  Min.   :0.0000   Min.   :38.00
##  1st Qu.:1.0000   1st Qu.:51.00
##  Median :1.0000   Median :60.00
##  Mean   :0.8615   Mean   :60.15
##  3rd Qu.:1.0000   3rd Qu.:67.00
##  Max.   :1.0000   Max.   :82.00
##      logwbc           frac
##  Min.   :3.362   Min.   :0.0000
##  1st Qu.:3.643   1st Qu.:1.0000
##  Median :3.732   Median :1.0000
##  Mean   :3.769   Mean   :0.7538
##  3rd Qu.:3.875   3rd Qu.:1.0000
##  Max.   :4.954   Max.   :1.0000
##      logpbm          protein
##  Min.   :0.4771   Min.   : 0.000
##  1st Qu.:1.3617   1st Qu.: 0.000
##  Median :1.6232   Median : 1.000
##  Mean   :1.5497   Mean   : 3.615
##  3rd Qu.:1.8451   3rd Qu.: 4.000
##  Max.   :2.0000   Max.   :27.000
##      scalc
##  Min.   : 7.00
##  1st Qu.: 9.00
##  Median :10.00
##  Mean   :10.12
##  3rd Qu.:10.00
##  Max.   :18.00
```

which gives means and five-number summaries for each of the

variables (the numeric ones, but they all are here, even the ones coded as 0 or 1 that are really categorical).

(b) Create a suitable response variable for a Cox proportional-hazards survival model, bearing in mind that the "event" here is death. Display your response variable, and explain briefly what the + signs attached to some of the variables mean, without using a technical term.

Solution

I seem to call my response variables y, but you can call yours whatever you like. Two things: the survival times, here time, and the indicator of the event, here vstatus being 1:

```
y = with(myeloma, Surv(time, vstatus == 1))
y
```

```
##  [1]  1.25   1.25   2.00   2.00   2.00
##  [6]  3.00   5.00   5.00   6.00   6.00
## [11]  6.00   6.00   7.00   7.00   7.00
## [16]  9.00  11.00  11.00  11.00  11.00
## [21] 11.00  13.00  14.00  15.00  16.00
## [26] 16.00  17.00  17.00  18.00  19.00
## [31] 19.00  24.00  25.00  26.00  32.00
## [36] 35.00  37.00  41.00  41.00  51.00
## [41] 52.00  54.00  58.00  66.00  67.00
## [46] 88.00  89.00  92.00   4.00+  4.00+
## [51]  7.00+  7.00+  8.00+ 12.00+ 11.00+
## [56] 12.00+ 13.00+ 16.00+ 19.00+ 19.00+
## [61] 28.00+ 41.00+ 53.00+ 57.00+ 77.00+
```

Or use myeloma$ (twice) before the variable names.

The values of y that have a + by them go with patients who were never observed to die (or were still alive at the end of the study). There were 17 of these, listed at the end of the data frame. Usually, these values of the response will be higher than the others, but they weren't here. (Maybe some of these patients were withdrawn from the study, or they joined it late.)

(c) What is the technical term for those patients that have a + by their values for the response variable?

Solution

Censored. A quick one mark. I was trying to dissuade you from using the word "censored" in the previous part, since I wanted you to demonstrate that you understood what it *meant*. But you should

know the technical term as well, which is why I asked you for it here. Grading note: if this part and the previous one contain, somewhere, the word "censored" *and* a clear explanation of what "censored" means, then I don't mind what is where.

(d) Fit a Cox proportional-hazards survival model predicting your response variable from all the other variables (except for the ones that you used to make the response variable). Display the `summary` of your model.

Solution

The obvious way to do this is to list all the other variables on the right side of the squiggle, but a faster way is this:

```
y.1 = coxph(y ~ . - time - vstatus, data = myeloma)
summary(y.1)
```

```
## Call:
## coxph(formula = y ~ . - time - vstatus, data = myeloma)
##
##   n= 65, number of events= 48
##
##              coef exp(coef) se(coef)      z
## logbun    1.85557   6.39536  0.65628  2.827
## hgb      -0.12629   0.88136  0.07212 -1.751
## platelet -0.25488   0.77501  0.51194 -0.498
## age      -0.01306   0.98702  0.01957 -0.668
## logwbc    0.35389   1.42460  0.71576  0.494
## frac      0.34232   1.40821  0.40725  0.841
## logpbm    0.38165   1.46470  0.48743  0.783
## protein   0.01302   1.01311  0.02612  0.498
## scalc     0.12976   1.13856  0.10502  1.236
##          Pr(>|z|)
## logbun    0.00469 **
## hgb       0.07994 .
## platelet  0.61858
## age       0.50439
## logwbc    0.62101
## frac      0.40059
## logpbm    0.43364
## protein   0.61817
## scalc     0.21659
## ---
## Signif. codes:
##   0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1  ' ' 1
##
```

```
##          exp(coef) exp(-coef) lower .95
## logbun      6.3954     0.1564    1.7670
## hgb         0.8814     1.1346    0.7652
## platelet    0.7750     1.2903    0.2841
## age         0.9870     1.0131    0.9499
## logwbc      1.4246     0.7020    0.3503
## frac        1.4082     0.7101    0.6339
## logpbm      1.4647     0.6827    0.5634
## protein     1.0131     0.9871    0.9625
## scalc       1.1386     0.8783    0.9268
##          upper .95
## logbun      23.147
## hgb          1.015
## platelet     2.114
## age          1.026
## logwbc       5.794
## frac         3.128
## logpbm       3.808
## protein      1.066
## scalc        1.399
##
## Concordance= 0.675  (se = 0.051 )
## Rsquare= 0.237   (max possible= 0.991 )
## Likelihood ratio test= 17.62  on 9 df,   p=0.04
## Wald test            = 17.93  on 9 df,   p=0.04
## Score (logrank) test = 18.97  on 9 df,   p=0.03
```

The `.` in this model formula means "all the columns in the data frame" (except for the response variable if it was in the data frame, which here it was not). I used `time` and `vstatus` to make `y`, so I had to exclude them explicitly.

If you forget to exclude `time` and `vstatus`, you are in danger of having a model that fits perfectly:

```
y.00 = coxph(y ~ ., data = myeloma)
```

```
## Warning in fitter(X, Y, strats, offset,
## init, control, weights = weights, :
## Loglik converged before variable
## 1,2,3,4,5,6,7,8,9,10,11 ; beta may be
## infinite.
```

```
summary(y.00)
```

```
## Call:
## coxph(formula = y ~ ., data = myeloma)
```

```
##
##   n= 65, number of events= 48
##
##                coef  exp(coef)   se(coef)
## time     -9.889e+00  5.071e-05  1.824e+01
## vstatus   1.439e+01  1.775e+06  3.703e+02
## logbun    4.864e-04  1.000e+00  8.626e-01
## hgb       2.776e-05  1.000e+00  1.417e-01
## platelet  2.287e-04  1.000e+00  7.862e-01
## age      -1.144e-05  1.000e+00  2.834e-02
## logwbc   -6.984e-06  1.000e+00  1.121e+00
## frac     -1.930e-04  9.998e-01  9.787e-01
## logpbm    8.106e-04  1.001e+00  9.518e-01
## protein   1.018e-04  1.000e+00  8.550e-02
## scalc    -4.515e-05  1.000e+00  1.639e-01
##               z Pr(>|z|)
## time     -0.542    0.588
## vstatus   0.039    0.969
## logbun    0.001    1.000
## hgb       0.000    1.000
## platelet  0.000    1.000
## age       0.000    1.000
## logwbc    0.000    1.000
## frac      0.000    1.000
## logpbm    0.001    0.999
## protein   0.001    0.999
## scalc     0.000    1.000
##
##          exp(coef) exp(-coef)  lower .95
## time     5.071e-05  1.972e+04  1.500e-20
## vstatus  1.775e+06  5.632e-07 1.209e-309
## logbun   1.000e+00  9.995e-01  1.845e-01
## hgb      1.000e+00  1.000e+00  7.576e-01
## platelet 1.000e+00  9.998e-01  2.142e-01
## age      1.000e+00  1.000e+00  9.460e-01
## logwbc   1.000e+00  1.000e+00  1.111e-01
## frac     9.998e-01  1.000e+00  1.468e-01
## logpbm   1.001e+00  9.992e-01  1.550e-01
## protein  1.000e+00  9.999e-01  8.458e-01
## scalc    1.000e+00  1.000e+00  7.252e-01
##          upper .95
## time     1.714e+11
## vstatus        Inf
## logbun   5.425e+00
```

```
## hgb      1.320e+00
## platelet 4.670e+00
## age      1.057e+00
## logwbc   8.997e+00
## frac     6.808e+00
## logpbm   6.464e+00
## protein  1.183e+00
## scalc    1.379e+00
##
## Concordance= 1  (se = 0 )
## Rsquare= 0.986   (max possible= 0.991 )
## Likelihood ratio test= 277  on 11 df,   p=<2e-16
## Wald test            = 0.3  on 11 df,   p=1
## Score (logrank) test = 87.4  on 11 df,   p=5e-14
```

The warning at the top is your clue that something has gone wrong. This kind of warning *can* happen with real data, but not often: it is usually an indication that something is wrong with the way you specified the model. If you look at the output, you'll realize that predicting survival time from survival time makes no sense at all.

There is of course nothing wrong with typing out all the variable names, except that the first time you type them out, you will likely make a typo (unless you are more careful than I usually am).

(e) In your model, which explanatory variables have a P-value less than 0.10? Fit a model containing only those and display the results.

Solution

Only `logbun` and `hgb`; the other P-values are larger, usually much larger. Because there are so many variables to remove, I am frightened away from `update` here (which I would normally try to use in this situation). I'm going to copy-and-paste my code for `y.1` and edit it:

```
y.2 = coxph(y ~ logbun + hgb, data = myeloma)
summary(y.2)
```

```
## Call:
## coxph(formula = y ~ logbun + hgb, data = myeloma)
##
##   n= 65, number of events= 48
##
##            coef exp(coef) se(coef)      z
## logbun  1.71597   5.56209  0.61855  2.774
```

```
## hgb    -0.11966   0.88722  0.05742 -2.084
##        Pr(>|z|)
## logbun  0.00553 **
## hgb     0.03717 *
## ---
## Signif. codes:
##   0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1  ' ' 1
##
##        exp(coef) exp(-coef) lower .95
## logbun    5.5621     0.1798    1.6547
## hgb       0.8872     1.1271    0.7928
##        upper .95
## logbun   18.6961
## hgb       0.9929
##
## Concordance= 0.675  (se = 0.043 )
## Rsquare= 0.172   (max possible= 0.991 )
## Likelihood ratio test= 12.27  on 2 df,   p=0.002
## Wald test            = 12.51  on 2 df,   p=0.002
## Score (logrank) test = 13.07  on 2 df,   p=0.001
```

That's all I wanted, but you can note that `hgb` has become significant at $\alpha = 0.05$. I suspect it was somewhat correlated with a variable that we removed, so that its value to the regression has become clearer.

(f) Do a test to compare the two models that you fit. Why do you prefer the second model? Explain briefly.

Solution

Comparing two models is `anova`, which also works here. The right `test` is `Chisq`:

```r
anova(y.2, y.1, test = "Chisq")
```

```
## Analysis of Deviance Table
##  Cox model: response is  y
##  Model 1: ~ logbun + hgb
##  Model 2: ~ (time + vstatus + logbun + hgb + platelet + age + logwbc + frac + logpbm + protein + scalc)
##    loglik Chisq Df P(>|Chi|)
## 1 -148.06
## 2 -145.39 5.347  7    0.6177
```

The usual logic here: this is far from significant, so the null hypothesis (that the two models are equally good) is not rejected, so we prefer the smaller model `y.2` because it is simpler.

I wasn't sure about the `Model 2` line of my `anova` output (what are `time` and `vstatus` doing there?), but the test has 7 degrees of freedom, which is correct since we started with 9 explanatory variables and finished with 2, so that we took out 7 of them. I checked what went off the right side of the page: there is a `-time-vstatus` on the end, so that it is correct. What happened is that the `.` got expanded out into all the variables separated by +, and then whatever else (the "minus" variables) were on the end.

In case you are curious, `step` also works on models like these:

```
y.3 = step(y.1, direction = "backward", trace = 0)
summary(y.3)
```

```
## Call:
## coxph(formula = y ~ logbun + hgb, data = myeloma)
##
##   n= 65, number of events= 48
##
##            coef exp(coef) se(coef)      z
## logbun  1.71597   5.56209  0.61855  2.774
## hgb    -0.11966   0.88722  0.05742 -2.084
##        Pr(>|z|)
## logbun  0.00553 **
## hgb     0.03717 *
## ---
## Signif. codes:
##   0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1  ' ' 1
##
##        exp(coef) exp(-coef) lower .95
## logbun    5.5621     0.1798    1.6547
## hgb       0.8872     1.1271    0.7928
##        upper .95
## logbun   18.6961
## hgb       0.9929
##
## Concordance= 0.675  (se = 0.043 )
## Rsquare= 0.172   (max possible= 0.991 )
## Likelihood ratio test= 12.27  on 2 df,   p=0.002
## Wald test            = 12.51  on 2 df,   p=0.002
## Score (logrank) test = 13.07  on 2 df,   p=0.001
```

The same model as the one we found by brute force. You can change the value of `trace` to see the progress, but in this case it's not very illuminating, since `<none>` and the variables we end up keeping are always at the bottom of the list to remove.

`step` is built on `add1` and `drop1`. In this case, `drop1` is run repeatedly and the variable with lowest AIC is removed. We had all numeric variables in this one, but if our model had something categorical like `treatment` with, let's say, 4 levels, `drop1` would contemplate dropping all four of these in one shot, the same way it works with a categorical variable in a regression of any other kind.

(g) There should be two explanatory variables left in your model. These are both numerical variables. Find their first and third quartiles, any way you like.

Solution

The obvious way is probably this:

```
quantile(myeloma$logbun)
```

```
##     0%    25%    50%    75%   100%
## 0.7782 1.1461 1.3222 1.5682 2.2355
```

```
quantile(myeloma$hgb)
```

```
##   0%  25%  50%  75% 100%
##  4.9  8.8 10.2 12.0 14.6
```

So the quartiles are 1.15 and 1.57 for `logbun`, and 8.8 and 12.0 for `hgb`.

There are (at least) three other ways to do it. This is the easiest:

```
summary(myeloma)
```

```
##       time          vstatus
##  Min.   : 1.25   Min.   :0.0000
##  1st Qu.: 7.00   1st Qu.:0.0000
##  Median :15.00   Median :1.0000
##  Mean   :24.01   Mean   :0.7385
##  3rd Qu.:35.00   3rd Qu.:1.0000
##  Max.   :92.00   Max.   :1.0000
##      logbun            hgb
##  Min.   :0.7782   Min.   : 4.9
##  1st Qu.:1.1461   1st Qu.: 8.8
##  Median :1.3222   Median :10.2
##  Mean   :1.3929   Mean   :10.2
##  3rd Qu.:1.5682   3rd Qu.:12.0
##  Max.   :2.2355   Max.   :14.6
##     platelet           age
##  Min.   :0.0000   Min.   :38.00
##  1st Qu.:1.0000   1st Qu.:51.00
```

```
##  Median :1.0000   Median :60.00
##  Mean   :0.8615   Mean   :60.15
##  3rd Qu.:1.0000   3rd Qu.:67.00
##  Max.   :1.0000   Max.   :82.00
##      logwbc           frac
##  Min.   :3.362   Min.   :0.0000
##  1st Qu.:3.643   1st Qu.:1.0000
##  Median :3.732   Median :1.0000
##  Mean   :3.769   Mean   :0.7538
##  3rd Qu.:3.875   3rd Qu.:1.0000
##  Max.   :4.954   Max.   :1.0000
##      logpbm          protein
##  Min.   :0.4771   Min.   : 0.000
##  1st Qu.:1.3617   1st Qu.: 0.000
##  Median :1.6232   Median : 1.000
##  Mean   :1.5497   Mean   : 3.615
##  3rd Qu.:1.8451   3rd Qu.: 4.000
##  Max.   :2.0000   Max.   :27.000
##      scalc
##  Min.   : 7.00
##  1st Qu.: 9.00
##  Median :10.00
##  Mean   :10.12
##  3rd Qu.:10.00
##  Max.   :18.00
```

from which you pick out the ones you need. Or, you `select` the ones you need first:

```
myeloma %>% select(logbun, hgb) %>% summary()
```

```
##      logbun            hgb
##  Min.   :0.7782   Min.   : 4.9
##  1st Qu.:1.1461   1st Qu.: 8.8
##  Median :1.3222   Median :10.2
##  Mean   :1.3929   Mean   :10.2
##  3rd Qu.:1.5682   3rd Qu.:12.0
##  Max.   :2.2355   Max.   :14.6
```

The `with` method is a bit hairy because there are two things to calculate inside, so you have to do it like this:

```
with(myeloma, {
    print(quantile(logbun))
    print(quantile(hgb))
})
```

```
##     0%    25%    50%    75%   100%
## 0.7782 1.1461 1.3222 1.5682 2.2355
##   0%  25%  50%  75% 100%
##  4.9  8.8 10.2 12.0 14.6
```

with the same result. Note that the multiple things to do are surrounded by curly brackets, and to display something within curly brackets you have to explicitly `print` it.

The `tidyverse` way is actually a bit inelegant, because you have to calculate two things for two variables (`summarize` will only allow you to have a single-number answer):

```
myeloma %>% summarize(logbun.q1 = quantile(logbun,
    0.25), logbun.q3 = quantile(logbun, 0.75),
    hgb.q1 = quantile(hgb, 0.25), hgb.q3 = quantile(hgb,
        0.75))
```

```
## # A tibble: 1 x 4
##   logbun.q1 logbun.q3 hgb.q1 hgb.q3
##       <dbl>     <dbl>  <dbl>  <dbl>
## 1      1.15      1.57    8.8     12
```

There is a `tidyverse` way to obtain the quartiles of *all* the columns. It lives in package `purrr`, which gets loaded with the `tidyverse`:

```
map_dbl(myeloma, quantile, 0.25)
```

```
##     time  vstatus   logbun      hgb platelet
##   7.0000   0.0000   1.1461   8.8000   1.0000
##      age   logwbc     frac   logpbm  protein
##  51.0000   3.6435   1.0000   1.3617   0.0000
##    scalc
##   9.0000
```

```
map_dbl(myeloma, quantile, 0.75)
```

```
##     time  vstatus   logbun      hgb platelet
##  35.0000   1.0000   1.5682  12.0000   1.0000
##      age   logwbc     frac   logpbm  protein
##  67.0000   3.8751   1.0000   1.8451   4.0000
##    scalc
##  10.0000
```

So what you do is a `select` to get just `logbun` and `hgb`, first, and then, for example:

```
myeloma %>% select(logbun, hgb) %>% map_dbl(quantile,
    0.25)
```

```
## logbun    hgb
## 1.1461 8.8000
```

Can we get them both in one shot? Let's see what this does:

```
myeloma %>% select(logbun, hgb) %>% map(quantile,
    c(0.25, 0.75))
```

```
## $logbun
##    25%    75%
## 1.1461 1.5682
##
## $hgb
##  25%  75%
##  8.8 12.0
```

This is what R calls a `list`. It would look nicer as a data frame. There is a way to get that:

```
myeloma %>% select(logbun, hgb) %>% map_df(quantile,
    c(0.25, 0.75))
```

```
## # A tibble: 2 x 2
##   logbun   hgb
##    <dbl> <dbl>
## 1   1.15   8.8
## 2   1.57  12
```

The `25%` and `75%` have gone missing. They were actually the `names` of each of the components of the list, which I don't think we easily have access to.

The way, as we have seen elsewhere, is to use *tidy(quantile)* or *enframe(quantile)*, which produce a two-column data frame with the percentiles shown.

(h) Create a data frame containing all possible combinations of the two quartiles for each of the two variables, and display the result.

Solution

The usual `crossing`:

```
logbuns = c(1.1461, 1.5682)
hgbs = c(8.8, 12)
new = crossing(logbun = logbuns, hgb = hgbs)
new
```

```
## # A tibble: 4 x 2
##   logbun   hgb
##    <dbl> <dbl>
## 1   1.15   8.8
## 2   1.15  12
## 3   1.57   8.8
## 4   1.57  12
```

Or anything equivalent to that. This is my procedure, but you are free to shorten it if it will still work. The place you have to get to in the end is a data frame with columns called `logbun` and `hgb`, and the right four combinations of values. If you want to round them off, that's fine; it won't affect the graph that's coming up.

(i) Obtain predicted survival probabilities for each of the combinations of variables you created above. You don't need to look at the results (they are rather long).

Solution

This seems as if it ought to be `predict`, but the `survival` version of it is called `survfit`:

```
s = survfit(y.2, new, data = myeloma)
```

It works the same as `predict`: a fitted model object (your smaller survival model), and a data frame of values to predict for. The `data=` is not strictly needed here, but if you want `ggsurvplot` to work right, you *do* need it to be here.

(j) Obtain a graph of the predicted survival curves for each combination of your variables.

Solution

This is easier than you think: it's just `ggsurvplot` from `survminer`:

```
ggsurvplot(s, conf.int = F)
```

(k) Is it better to have high or low values for each of the variables in your prediction? Explain briefly.

Solution

Those four "strata" are the four rows in your prediction data frame (the four combinations). They are in the same order that they were in `new` (or whatever name you used):

```
new
```

```
## # A tibble: 4 x 2
##   logbun   hgb
##    <dbl> <dbl>
## 1   1.15   8.8
## 2   1.15  12
## 3   1.57   8.8
## 4   1.57  12
```

The best survival curve is the top-right green one. This is stratum 2, from the legend at the top. In texttt{new}, this goes with a low value of `logbun` and a *high* value of `hgb`.

Strata is plural; the singular is *stratum*. Like data and datum.

You can check this by looking at the *worst* survival curve, which should be diametrically opposed. This is the blue one, stratum 3, which is *high* `logbun` and *low* `hgb`, indeed exactly the opposite of the best one.

Things that are tests, like `logbun`, are often set up so that a high value is the abnormal one (so that an abnormal one will be easy to spot). Things that are measurements, like `hgb`, might have an ideal range, but the better value could be high or low, depending on what is being measured.

18.4 Ovarian cancer

R's `survival` package contains several data sets. One of these is called `ovarian`; it comes from a study of 26 ovarian cancer patients. The major purpose of this study was to compare the effects of two treatments on survival time.

(a) Obtain and display (all of) the data set. This is as simple as loading the package and typing the data set's name.

Solution

Thus. You may need to start with `library(survival)`:

```
ovarian
```

```
##    futime fustat     age resid.ds rx ecog.ps
## 1      59      1 72.3315        2  1       1
## 2     115      1 74.4932        2  1       1
## 3     156      1 66.4658        2  1       2
## 4     421      0 53.3644        2  2       1
## 5     431      1 50.3397        2  1       1
## 6     448      0 56.4301        1  1       2
## 7     464      1 56.9370        2  2       2
```

```
## 8     475      1 59.8548        2  2       2
## 9     477      0 64.1753        2  1       1
## 10    563      1 55.1781        1  2       2
## 11    638      1 56.7562        1  1       2
## 12    744      0 50.1096        1  2       1
## 13    769      0 59.6301        2  2       2
## 14    770      0 57.0521        2  2       1
## 15    803      0 39.2712        1  1       1
## 16    855      0 43.1233        1  1       2
## 17   1040      0 38.8932        2  1       2
## 18   1106      0 44.6000        1  1       1
## 19   1129      0 53.9068        1  2       1
## 20   1206      0 44.2055        2  2       1
## 21   1227      0 59.5890        1  2       2
## 22    268      1 74.5041        2  1       2
## 23    329      1 43.1370        2  1       1
## 24    353      1 63.2192        1  2       2
## 25    365      1 64.4247        2  2       1
## 26    377      0 58.3096        1  2       1
```

There are indeed 26 rows. This is a `data.frame` rather than a `tibble`, so you see the whole thing, in case you were expecting something like this:

```
ovarian %>% as.tibble()
```

```
## # A tibble: 26 x 6
##    futime fustat   age resid.ds    rx ecog.ps
##  *  <dbl>  <dbl> <dbl>    <dbl> <dbl>   <dbl>
##  1     59      1  72.3        2     1       1
##  2    115      1  74.5        2     1       1
##  3    156      1  66.5        2     1       2
##  4    421      0  53.4        2     2       1
##  5    431      1  50.3        2     1       1
##  6    448      0  56.4        1     1       2
##  7    464      1  56.9        2     2       2
##  8    475      1  59.9        2     2       2
##  9    477      0  64.2        2     1       1
## 10    563      1  55.2        1     2       2
## # ... with 16 more rows
```

which doesn't change anything in `ovarian`, but changes what kind of thing it is (and thus how it displays). Usually when you read something in from a file, you use something like `read_delim` that makes a `tibble`, but this one wasn't read in from a file. It was stored

in the package as an old-fashioned `data.frame`, and so that's how it stays.

(b) The columns of interest to us are:

- `futime`: the time for which a patient was followed-up: the number of days until either they died or the study ended (or they withdrew from the study for some other reason).
- `fustat`: follow-up status: 1 if the patient died of ovarian cancer, 0 if they were still alive when the study ended.
- `age`: of patient, at diagnosis, in years
- `rx`: treatment, numbered 1 or 2, but really labels for the two treatments.

Create and display a suitable response variable `y` for a Cox proportional-hazards model.

Solution

The idea is to use the appropriate one(s) of these columns in `Surv`. Remember that the response variable in a survival model encodes two things: the survival time, and whether or not the event (here death) actually happened to that patient or not. I always forget whether the second thing in `Surv` has to be 1 or 0 if the event happened. The help says that it needs to be 1 or `TRUE` if the event (death) happened, which is what `fustat` is, so we can use it as it is:

```
y = with(ovarian, Surv(futime, fustat))
y
```

```
##  [1]   59   115   156   421+  431   448+
##  [7]  464   475   477+  563   638   744+
## [13]  769+  770+  803+  855+ 1040+ 1106+
## [19] 1129+ 1206+ 1227+  268   329   353
## [25]  365   377+
```

This creates a separate variable `y` outside of any data frame. This is how it has to be done, because although `y` looks as if it is 26 long (one per patient), it's actually more complicated than that:

```
ov2 = ovarian %>% mutate(y = Surv(futime, fustat))
```

```
## Error in mutate_impl(.data, dots): Column ‘y‘ must be length 26 (the number of rows) or one, not 52
```

This doesn't work, because internally `y` is a `matrix` with two columns:

```
print.default(y)
```

```
##       time status
##  [1,]   59      1
##  [2,]  115      1
##  [3,]  156      1
##  [4,]  421      0
##  [5,]  431      1
##  [6,]  448      0
##  [7,]  464      1
##  [8,]  475      1
##  [9,]  477      0
## [10,]  563      1
## [11,]  638      1
## [12,]  744      0
## [13,]  769      0
## [14,]  770      0
## [15,]  803      0
## [16,]  855      0
## [17,] 1040      0
## [18,] 1106      0
## [19,] 1129      0
## [20,] 1206      0
## [21,] 1227      0
## [22,]  268      1
## [23,]  329      1
## [24,]  353      1
## [25,]  365      1
## [26,]  377      0
## attr(,"type")
## [1] "right"
## attr(,"class")
## [1] "Surv"
```

The way R works out the length of something is its total number of entries, which for this matrix is $26 \times 2 = 52$. The only things you can add via `mutate` to a data frame with 26 rows have length 26 (since everything in a data frame has to be the same length) or length 1 (in which case it is repeated 26 times).

With some cajoling, you can get this into a data frame, but with more cajoling than seems necessary. It's much easier to leave it as a separate thing outside a data frame.

I gave `ov2` a "temporary" name because I am not going to use it again.

In some respects, it's like that two-column response in logistic regression, where each row of the data frame represents several observations pooled together.

(c) In the display of your response variable, some values are marked with a +. Why is that? Explain briefly. (If you use a technical term,

you should explain what it means.)

Solution

These are the censored observations. You can say this, but you also need to say what that means (this is the "technical term" referred to in the question). The observations with a + are individuals who were never observed to die, or who were still alive at the end of the study. I want you to demonstrate that you know what censored *means*, not just that you know when you have a censored observation. Extra: in a study like this, patients are typically "recruited" into the study at various different times. Patients who happened to be in the study near the beginning and who survived can have a large (censored) value of `y` (like those values over 1000 days). But a patient might join the study later on; if they survive, they might produce a censored observation with a small survival time, like the last value 377. I'm sure the doctor would have liked to follow them for longer, but the funding ran out, and the doctor had a paper to write. (There is *some* information in these small censored values, but not much, because most of the patients, even the ones who eventually died, survived for longer than 377 days.) The other thing that might have happened is that a patient with the 377-censored value died *from something else* unrelated to ovarian cancer. The study is only concerned with deaths from ovarian cancer, so such a patient is treated as censored at their death time. After this point we cannot assess how long this patient survived *ovarian cancer*.

(d) Fit a Cox proportional-hazards model for predicting survival time from age and treatment. Note that the numeric values for treatment make sense only as labels for the two treatments, so in your model formula make treatment into a factor. Display the results.

Solution

The hint suggests something like this:

```
time.1 = coxph(y ~ age + factor(rx), data = ovarian)
summary(time.1)
```

```
## Call:
## coxph(formula = y ~ age + factor(rx), data = ovarian)
##
##   n= 26, number of events= 12
##
##                  coef exp(coef) se(coef)
## age          0.14733   1.15873  0.04615
## factor(rx)2 -0.80397   0.44755  0.63205
```

```
##                  z Pr(>|z|)
## age          3.193  0.00141 **
## factor(rx)2 -1.272  0.20337
## ---
## Signif. codes:
##   0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1  ' ' 1
##
##             exp(coef) exp(-coef) lower .95
## age            1.1587      0.863    1.0585
## factor(rx)2    0.4475      2.234    0.1297
##             upper .95
## age             1.268
## factor(rx)2     1.545
##
## Concordance= 0.798  (se = 0.076 )
## Rsquare= 0.457   (max possible= 0.932 )
## Likelihood ratio test= 15.89  on 2 df,   p=4e-04
## Wald test            = 13.47  on 2 df,   p=0.001
## Score (logrank) test = 18.56  on 2 df,   p=9e-05
```

Alternatively, define the factor version of rx in the data frame first. This is the slick way to do that:

```
time.1a = ovarian %>% mutate(rxf = factor(rx)) %>%
    coxph(y ~ age + rxf, data = .)
summary(time.1a)
```

```
## Call:
## coxph(formula = y ~ age + rxf, data = .)
##
##   n= 26, number of events= 12
##
##          coef exp(coef) se(coef)      z
## age   0.14733   1.15873  0.04615  3.193
## rxf2 -0.80397   0.44755  0.63205 -1.272
##      Pr(>|z|)
## age   0.00141 **
## rxf2  0.20337
## ---
## Signif. codes:
##   0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1  ' ' 1
##
##      exp(coef) exp(-coef) lower .95
## age     1.1587      0.863    1.0585
## rxf2    0.4475      2.234    0.1297
```

```
##      upper .95
## age      1.268
## rxf2     1.545
##
## Concordance= 0.798  (se = 0.076 )
## Rsquare= 0.457   (max possible= 0.932 )
## Likelihood ratio test= 15.89  on 2 df,   p=4e-04
## Wald test            = 13.47  on 2 df,   p=0.001
## Score (logrank) test = 18.56  on 2 df,   p=9e-05
```

The answer is the same either way. The `data=.` means "use the data frame that came out of the previous step, the one with `rxf` in it."

(e) Is there a significant difference between the treatments in terms of their effects on survival (from ovarian cancer)?

Solution

Look at the P-value for my `factor(rx)2`, 0.203. This is not small, so there is no evidence of a difference between treatments. Extra: the reason for the odd label is that we have turned treatment into a categorical variable; treatment 1 is used as the baseline, and the negative slope says that the "hazard of death" is lower for treatment 2 than for treatment 1: that is, people survive longer on treatment 2, but the difference is not big enough to be significant (we also have a smallish sample size). Since there are only two treatments, it would in fact have been OK to leave them as numbers (with two numbers one unit apart, the slope would have been the same size as here), but I think it's a good idea to treat categorical variables as categorical. My own habit is to use letters or something non-numerical to distinguish categories. I might have used `t1` and `t2` in this case, or the names of the different treatments.

(f) Is there a significant effect of age? If there is, describe the effect that age has on survival.

Solution

The P-value for age is 0.0014, small, so age definitely has a significant effect on survival. As to what kind of effect, look at the slope coefficient, 0.15, positive, which means that increased age-at-diagnosis goes with an *increased* hazard of death, or that older patients do not survive as long. I would like you to get to the plain-English words at the end. Part of your job as a statistician is explaining what you got to people who are doctors, managers, etc., who won't understand the terminology. Thus, one mark for assessing significance via P-value, one for looking at the slope coefficient and noting that it is positive, and one for getting to "older patients do not survive as long", or

"older patients have a larger chance of dying sooner". (Strictly, this is also "all else equal" as usual, since survival time might also have depended on treatment, but the point of this question is for you to get to "older patients do not survive as long".) (The interpretation of the slope may seem backwards: a positive slope means a *shorter* survival time for a larger age. This is why I talk about "hazard of death", since that guides us to the correct interpretation.) Extra: I was curious about what would happen if I just included `rx` in the model:

```
time.2 = update(time.1, . ~ . - age)
summary(time.2)
```

```
## Call:
## coxph(formula = y ~ factor(rx), data = ovarian)
##
##   n= 26, number of events= 12
##
##                coef exp(coef) se(coef)
## factor(rx)2 -0.5964    0.5508   0.5870
##                  z Pr(>|z|)
## factor(rx)2 -1.016     0.31
##
##             exp(coef) exp(-coef) lower .95
## factor(rx)2    0.5508      1.816    0.1743
##             upper .95
## factor(rx)2      1.74
##
## Concordance= 0.608  (se = 0.07 )
## Rsquare= 0.04   (max possible= 0.932 )
## Likelihood ratio test= 1.05  on 1 df,   p=0.3
## Wald test            = 1.03  on 1 df,   p=0.3
## Score (logrank) test = 1.06  on 1 df,   p=0.3
```

Still not significant, but this model is a *lot worse* because I took out the significant `age`. What this is doing is mixing up all the people of different ages (and we know that age has an effect on survival) and trying (and failing) to discern an effect of treatment.

We could have been badly misled by this model if one of the treatments had predominantly older patients. We know that older patients have worse survival, so the treatment with older patients would have looked worse, even if it actually wasn't. The model `time.1` which contained `age` properly adjusted for the effect of age, so that was the best way to see whether there was a difference between treatments.

What you often see early on in a paper on this kind of stuff is

a graph showing that the treatment groups are similar in terms of important things like age. Here, that could be a boxplot:

```
ggplot(ovarian, aes(x = factor(rx), y = age)) +
    geom_boxplot()
```

I needed to do factor(rx) because geom_boxplot needs a genuine categorical variable, not just a numerical variable masquerading as one. If you just leave it as rx, as I discovered, you get *one* boxplot of all the ages together regardless of treatment. The key, for you as user of software, is not (necessarily) to get it right the first time, but to know what to do to fix up the errors you will inevitably get. If you have worked through the boxplot examples in C32 and D29, you will have enough experience to remember that a boxplot has to have a categorical x (text will do, but definitely not numbers). This is why I give you so many things to work through: so that you gain the experience to know how to fix up problems.

Treatment 1 has a larger spread of ages and treatment 2 has a low outlier age, but the median ages are very similar.

(g) Make a martingale residual plot for this model. Do you see any problems? Explain briefly.

Solution

The plot is just the same idea as the one in the notes. Make sure you have survminer installed and loaded:

```
ggcoxdiagnostics(time.1) + geom_smooth()
```

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

Make a call about whether you think the smooth trend deviates too much from the red dotted line going across at zero. Martingale residuals can get very negative (and that is OK), so that residual of -2 is not a problem, and this is pulling the smooth trend down a bit (it's the reason for the dip on the right side of the smooth trend). So I'd call this OK, but you can make whatever call you like as long as it's supported by what you see here.

I observe that the obvious fixable thing is where you have a curve here, one that looks like a parabola (at which point you add squared terms to your explanatory variables and see if that helps, as for `bmi` in one of the other problems). This one is too wiggly to be a parabola (it bends *twice*), and so is more like a cubic than anything.

The other thing you can note is that the grey envelope is "not significantly different from zero", since 0 is clearly within the grey envelope all the way across.

(h) Find the quartiles of `age`, and make a data frame containing all combinations of those two ages and the two treatments. Display what you have. (Feel free to copy the values by hand, rather than trying to save them and use them.)

Solution

I imagine you can guess what we are going to be doing with these predictions, so we'll call the data frame `new` when we get there. Quartiles first:

```
quantile(ovarian$age)
```

```
##       0%      25%      50%      75%     100%
## 38.89320 50.16712 56.84660 62.37810 74.50410
```

or, if you prefer,

```
ovarian %>% summarize(q1 = quantile(age, 0.25),
    q3 = quantile(age, 0.75))
```

```
##         q1      q3
## 1 50.16712 62.3781
```

The quartiles are 50.17 and 62.38 (rounding slightly).

Either way is good.

Then follow my standard procedure (or one of your own devising), remembering that "treatment" is called `rx` here:

```
ages = c(50.17, 62.38)
rxs = c(1, 2)
new = crossing(age = ages, rx = rxs)
new
```

```
## # A tibble: 4 x 2
##     age    rx
##   <dbl> <dbl>
## 1  50.2     1
## 2  50.2     2
## 3  62.4     1
## 4  62.4     2
```

(i) Obtain predicted survival probabilities for each of your age-treatment combinations, for each of a variety of survival times. (This is only one thing, despite it sounding like a lot.)

Solution

The magic word here is `survfit`. The structure is the same as for `predict`:

```
s = survfit(time.1, new, data = ovarian)
summary(s)
```

```
## Call: survfit(formula = time.1, newdata = new, data = ovarian)
##
##  time n.risk n.event survival1 survival2
##    59     26       1     0.993     0.997
##   115     25       1     0.985     0.993
##   156     24       1     0.973     0.988
##   268     23       1     0.959     0.981
##   329     22       1     0.932     0.969
##   353     21       1     0.905     0.956
##   365     20       1     0.877     0.943
##   431     17       1     0.843     0.926
##   464     15       1     0.805     0.908
```

```
##   475     14       1     0.768     0.888
##   563     12       1     0.701     0.853
##   638     11       1     0.634     0.816
##  survival3 survival4
##      0.959     0.981
##      0.911     0.959
##      0.846     0.928
##      0.777     0.893
##      0.653     0.826
##      0.548     0.764
##      0.452     0.701
##      0.356     0.630
##      0.271     0.557
##      0.202     0.489
##      0.117     0.382
##      0.064     0.292
```

I didn't ask you to display it, so doing so is optional. Also, you don't need (here) that `data=ovarian` (the predictions will work just fine without it), but the plot coming up *will not*. So my recommendation is to put it in.

Extra: The four columns `survival1` through `survival4` are the survival probabilities for the times shown in the left-hand column (these are numbers of days), for the four *rows* of my `new`, in the same order.

There are only a few different times, because these are the numbers of days at which somebody in the data set died, and the estimated survival probability does not change at the times in between these. (You'll see this on a plot in a minute.)

These survival probabilities are pretty easy to eyeball: the best survival is in stratum 2, which is the younger patients in treatment 2. This we'll come back to.

(j) Draw a plot that compares the survival probabilities at the different times.

Solution

Thus. The `conf.int=F` means to skip the confidence interval "envelopes" that I find make the plot rather messy:

```
ggsurvplot(s, conf.int = F)
```

(k) According to your plot, how would you describe the effects of treatment and of age?

Solution

The best survival curve, in terms of surviving longest, is up and to the right, so the green one is best and the blue one is worst. To figure out which those are, we have to go back to the data frame we created:

In other cases, having the event happen sooner is better. For example, you might be poisoning rats, in which case you want them to die quicker. Or the event might be something desirable like becoming qualified to fly an airplane. In those cases, down and to the left is better.

```
new
```

```
## # A tibble: 4 x 2
##     age    rx
##   <dbl> <dbl>
## 1  50.2     1
## 2  50.2     2
## 3  62.4     1
## 4  62.4     2
```

To see the effect of age, compare strata 1 and 3 (or 2 and 4). This means comparing the red and blue curves; the red one is clearly better (in the sense of longer survival time), which means that age has a big effect on survival, with younger people living longer, other things being equal (that we saw earlier was significant). You could equally well compare the green and purple survival curves and come to the same conclusion.

To assess the effect of treatment, compare strata 1 and 2 (red and green), or strata 3 and 4 (blue and purple). In both cases, the stratum corresponding to treatment 2 has slightly better survival (has a higher chance of living for longer), but there is not as big an effect as for age. (You'll recall that the treatment difference was not significant).

One point for stating or implying that you know which stratum is which, and one each for saying something about the effects of age and of treatment, including which one is better.

Extra: recall the output from the Cox model:

```r
summary(time.1)
```

```
## Call:
## coxph(formula = y ~ age + factor(rx), data = ovarian)
##
##   n= 26, number of events= 12
##
##                 coef exp(coef) se(coef)
## age          0.14733   1.15873  0.04615
## factor(rx)2 -0.80397   0.44755  0.63205
##                  z Pr(>|z|)
## age          3.193  0.00141 **
## factor(rx)2 -1.272  0.20337
## ---
## Signif. codes:
##   0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1  ' ' 1
##
##             exp(coef) exp(-coef) lower .95
## age            1.1587      0.863    1.0585
## factor(rx)2    0.4475      2.234    0.1297
##             upper .95
## age             1.268
## factor(rx)2     1.545
##
## Concordance= 0.798  (se = 0.076 )
## Rsquare= 0.457   (max possible= 0.932 )
## Likelihood ratio test= 15.89  on 2 df,   p=4e-04
## Wald test            = 13.47  on 2 df,   p=0.001
## Score (logrank) test = 18.56  on 2 df,   p=9e-05
```

The slope coefficient for treatment 2 (as compared to the baseline treatment 1) was -0.83097, *negative*, which meant that patients on treatment 2 had a *lower* hazard of death than patients on treatment 1: that is, that treatment 2 was better for survival than treatment 1. That is what the plot said also (and the relatively small difference is consistent with that difference not being significant).

19
Analysis of Variance revisited

Packages for this chapter:

```
library(MASS)
library(car)

## Loading required package: carData

library(tidyverse)

## -- Attaching packages ---- tidyverse 1.2.1 --

## v ggplot2 3.1.0     v purrr   0.2.5
## v tibble  1.4.2     v dplyr   0.7.8
## v tidyr   0.8.1     v stringr 1.3.1
## v readr   1.1.1     v forcats 0.3.0

## -- Conflicts ------- tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
## x dplyr::recode() masks car::recode()
## x dplyr::select() masks MASS::select()
## x purrr::some()   masks car::some()
```

19.1 Focused comparisons of the effect of caffeine

Does caffeine help students do better on a certain test? To find out, 36 students were randomly allocated to three groups (12 in each group). Each student received a fixed number of cups of coffee while they were studying, but the students didn't know whether they were receiving all full-strength coffee ("high"), all decaf coffee ("low") or a 50-50 mixture of the two ("moderate"). For each subject, their group was recorded as well as their score on the test. The data are in link, as a `.csv` file.

This time we look at contrasts. Suppose I knew ahead of time that I wanted to compare moderate caffeine with high, and any caffeine with none. (In the latter case, we're comparing "no caffeine" against the average of the other two groups.)

In the previous go-through of the caffeine data, we discovered that `amount` was actually text rather than a factor, but we also discovered that it *didn't matter*. Here it does matter, so the first thing we have to do is to re-do the `gather`, creating a factor version of `amount`.

(a) Read in the data again, from link, and display it. This is the untidy format, so name it appropriately:

Solution

```
my_url = "http://individual.utoronto.ca/kbutler/stad29/caffeine.csv"
caffeine.untidy = read_csv(my_url)
```

```
## Parsed with column specification:
## cols(
##   Sub = col_integer(),
##   High = col_integer(),
##   Moderate = col_integer(),
##   None = col_integer()
## )
```

```
caffeine.untidy
```

```
## # A tibble: 12 x 4
##      Sub  High Moderate  None
##    <int> <int>    <int> <int>
##  1     1    72       68    68
##  2     2    65       80    74
##  3     3    68       64    59
##  4     4    83       65    61
##  5     5    79       69    65
##  6     6    92       79    72
##  7     7    69       80    80
##  8     8    74       63    58
##  9     9    78       69    65
## 10    10    83       70    60
## 11    11    88       83    78
## 12    12    71       75    75
```

One caffeine level per column, rather than a column of caffeine levels, so untidy.

(b) Copy your `gather` from before, only this time add `factor_key=T` to the end of it. Take a look at the results. What has changed from before?

Solution

We'll save into `caffeine` again:

```
caffeine = caffeine.untidy %>% gather(amount,
    score, High:None, factor_key = T)
caffeine
```

```
## # A tibble: 36 x 3
##      Sub amount score
##    <int> <fct>  <int>
##  1     1 High      72
##  2     2 High      65
##  3     3 High      68
##  4     4 High      83
##  5     5 High      79
##  6     6 High      92
##  7     7 High      69
##  8     8 High      74
##  9     9 High      78
## 10    10 High      83
## # ... with 26 more rows
```

The variable created for "how are the columns different" is now a `factor`: it was text before. Maybe we should have made it a factor before (it wouldn't have done any harm), but we got away with not doing so.

(c) Using the newly tidied caffeine data, run the ANOVA *as a regression* (that is, using `lm`). Look at the `summary` of the output. What do you see?

Solution

```
caffeine.2 = lm(score ~ amount, data = caffeine)
summary(caffeine.2)
```

```
##
## Call:
## lm(formula = score ~ amount, data = caffeine)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -11.833  -6.958  -2.458   6.354  15.167
##
## Coefficients:
##                Estimate Std. Error t value
```

```
## (Intercept)      76.833      2.235  34.383
## amountModerate   -4.750      3.160  -1.503
## amountNone       -8.917      3.160  -2.821
##                Pr(>|t|)
## (Intercept)     < 2e-16 ***
## amountModerate  0.14234
## amountNone      0.00803 **
## ---
## Signif. codes:
##   0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1  ' ' 1
##
## Residual standard error: 7.741 on 33 degrees of freedom
## Multiple R-squared:  0.1946, Adjusted R-squared:  0.1458
## F-statistic: 3.986 on 2 and 33 DF,  p-value: 0.02815
```

Look at the slopes. They are `amount` followed by one of the amounts of caffeine. R is using "high" as a baseline (that's the first level alphabetically), so the `amountModerate` line is testing high vs. moderate: high is *not* significantly higher, in terms of test scores, than moderate. That's one of the things I wanted to test. What about the coefficient for `amountNone`? That's none vs. *high*, since high was the baseline. This is, as we saw from Tukey, significant. But it is not what I said we wanted to test.

In case you're curious, you can also get the regular analysis of variance table as below. `anova` is multi-talented:

```r
anova(caffeine.2)
```

```
## Analysis of Variance Table
##
## Response: score
##           Df  Sum Sq Mean Sq F value  Pr(>F)
## amount     2  477.72 238.861  3.9861 0.02815
## Residuals 33 1977.50  59.924
##
## amount    *
## Residuals
## ---
## Signif. codes:
##   0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1  ' ' 1
```

The problem is that you can't naturally do Tukey this way, which is often what you want to do next. That's why we used `aov` before.

Since we have a regression model (albeit a peculiar one), we can test whether we should remove `amount` (that is, whether it has any impact on test scores) this way too:

```r
drop1(caffeine.2, test = "F")
```

```
## Single term deletions
##
## Model:
## score ~ amount
##        Df Sum of Sq    RSS    AIC F value
## <none>              1977.5 150.22
## amount  2    477.72 2455.2 154.01  3.9861
##         Pr(>F)
## <none>
## amount 0.02815 *
## ---
## Signif. codes:
##   0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1  ' ' 1
```

Same conclusion: there is some effect of caffeine level on test score.

(d) Obtain the different values of `amount`, in the order that R has them.

Solution
Count them, or find the distinct ones:

```r
caffeine %>% group_by(amount) %>% summarize(count = n())
```

```
## # A tibble: 3 x 2
##   amount   count
##   <fct>    <int>
## 1 High        12
## 2 Moderate    12
## 3 None        12
```

or there is this shortcut to the above:

```r
caffeine %>% count(amount)
```

```
## # A tibble: 3 x 2
##   amount       n
##   <fct>    <int>
## 1 High        12
## 2 Moderate    12
## 3 None        12
```

Or

```r
caffeine %>% distinct(amount)
```

```
## # A tibble: 3 x 1
##   amount
##   <fct>
## 1 High
## 2 Moderate
## 3 None
```

since we didn't really need to know how many of each there were.

These would all have worked if `amount` had been text rather a factor. If you have a genuine `factor`, you can also ask for its `levels`:

```
with(caffeine, levels(amount))
```

```
## [1] "High"     "Moderate" "None"
```

or `summary` will count them up and list them:

```
caffeine %>% select(amount) %>% summary()
```

```
##       amount
##  High    :12
##  Moderate:12
##  None    :12
```

This last won't work if you have a categorical-variable-as-text. It has to be a genuine factor for it to work.

The categories are High, Moderate and None in that order. For working with contrasts, we need to have the thing we're making contrasts for (see below) as a `factor`, otherwise it won't work.

(e) Create a contrast that compares High with Moderate, ignoring None. That is, create a vector whose length is the same as the number of levels of `amount`, and which has a 1 to represent High and a -1 to represent Moderate.

Solution

Put a 0 in for None:

```
c.hm = c(1, -1, 0)
```

Having the 1 and the -1 the other way around is also fine.

(f) Create a contrast that compares "any caffeine" against "none" by comparing None against the average of Moderate and High.

Solution

```
c.any = c(-0.5, -0.5, 1)
```

Note that both our contrasts have coefficients that add up to zero, as they must:

```
sum(c.hm)
```

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

```
sum(c.any)
```

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

(g) Verify that your two contrasts are orthogonal.

Solution

Multiply them together and check that what you get adds up to zero:

```
sum(c.hm * c.any)
```

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

Zero, so orthogonal. You can check that writing `c.any` as `c(-1,-1,2)` would also work (and still be orthogonal with `c.hm`), and so would writing it as `c(1,1,-2)`.

(h) Arrange your contrasts as columns of a matrix (using `cbind`), and say that you want to use these as contrasts for `amount` (in data frame `caffeine` or whatever you called it).

Solution

```
m = cbind(c.hm, c.any)
contrasts(caffeine$amount) = m
```

(i) Fit the ANOVA as an `lm`, and look at the `summary`. What do you conclude about your contrasts?

Solution

```
caff.3 = lm(score ~ amount, data = caffeine)
summary(caff.3)
```

```
##
## Call:
## lm(formula = score ~ amount, data = caffeine)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -11.833  -6.958  -2.458   6.354  15.167
```

```
##
## Coefficients:
##             Estimate Std. Error t value
## (Intercept)   72.278      1.290  56.022
## amountc.hm     2.375      1.580   1.503
## amountc.any   -4.361      1.825  -2.390
##             Pr(>|t|)
## (Intercept)   <2e-16 ***
## amountc.hm    0.1423
## amountc.any   0.0227 *
## ---
## Signif. codes:
##   0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1  ' ' 1
##
## Residual standard error: 7.741 on 33 degrees of freedom
## Multiple R-squared:  0.1946, Adjusted R-squared:  0.1458
## F-statistic: 3.986 on 2 and 33 DF,  p-value: 0.02815
```

`c.hm` was the contrast between high and moderate caffeine. This is not significant (P-value 0.142), which is the same conclusion as Tukey, but the P-value here is quite a bit lower (and thus closer to being significant). There's a reason for that: here we are focusing in on the two contrasts that we really wanted to test, and ignoring the *F*-test and the Tukey that tell us stuff that we don't care about. By focusing our comparison, we get a better (smaller) P-value.

`c.any` was none vs. average of any caffeine. This one is significant, with a P-value of 0.023. So this contrast tells us that having any caffeine is better than having none.

(j) What happens if you try to use high caffeine vs. moderate caffeine and moderate vs. none as your two contrasts?

Solution

```
c.hm = c(1, -1, 0)
c.mn = c(0, 1, -1)
```

So far so good: the coefficients add up to zero and they reflect the right comparisons. But now:

```
sum(c.hm * c.mn)
```

```
## [1] -1
```

This does *not* add up to zero, so these two contrasts are not orthogonal, and we can't do what we just did. R will give us an answer if we try it, but it'll be the *wrong* answer.

SAS, for example, has a way of making non-orthogonal contrasts orthogonal in a way that the user doesn't have to worry about, but in R, you are closer to the ground, so to speak, and you have to make it happen yourself.

The best description I have seen of what to do here is by David Howell , at link (at the bottom). Let me try to follow his method.

the author of a famous text on Statistics in Psychology.

First we need a vector that is all 1's, which I have called `c0` below. Since each of our contrasts `c.hm` and `c.mn` have 3 things in them (3 groups), we need to add a "dummy" 3rd contrast to give us a 3×3 array of numbers:

which we are going to invert, as a matrix. But I get ahead of myself.

```
c0 = rep(1, 3)
m = cbind(c0, c.hm, c.mn)
m
```

```
##      c0 c.hm c.mn
## [1,]  1    1    0
## [2,]  1   -1    1
## [3,]  1    0   -1
```

This is what Howell calls an "augmented" matrix of contrasts, since it has our two contrasts as the second and third columns, plus the extra dummy one. Next we invert this matrix of contrasts, which we can do because it's square. `t(m)` means "take the matrix transpose of `m`", if you're trying to keep up at the back, and `solve` finds a matrix inverse:

```
minv = solve(t(m))
```

and then we remove the first column, which represents the contrast that we didn't want anyway (what Howell calls "deaugmenting"):

```
m.contr = minv[, -1]
m.contr
```

```
##            c.hm       c.mn
## [1,]  0.6666667  0.3333333
## [2,] -0.3333333  0.3333333
## [3,] -0.3333333 -0.6666667
```

```
contrasts(caffeine$amount) = m.contr
```

The columns of `m.contr` are our new contrasts. Note that they appear to be something else: high vs. the average of moderate and none, and none vs. the average of moderate and high. Note that they are actually *not* orthogonal, but if Howell is to be trusted, they can be used to test what we want. I think Howell is famous enough to be trusted. Now fit the model again:

```
caff.4 = lm(score ~ amount, data = caffeine)
summary(caff.4)
```

```
##
## Call:
## lm(formula = score ~ amount, data = caffeine)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -11.833  -6.958  -2.458   6.354  15.167
##
## Coefficients:
##             Estimate Std. Error t value
## (Intercept)   72.278      1.290  56.022
## amountc.hm     4.750      3.160   1.503
## amountc.mn     4.167      3.160   1.318
##             Pr(>|t|)
## (Intercept)   <2e-16 ***
## amountc.hm     0.142
## amountc.mn     0.196
## ---
## Signif. codes:
##   0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1  ' ' 1
##
## Residual standard error: 7.741 on 33 degrees of freedom
## Multiple R-squared:  0.1946, Adjusted R-squared:  0.1458
## F-statistic: 3.986 on 2 and 33 DF,  p-value: 0.02815
```

The rows `amountc.hm` and `amountc.mn` are the proper tests for our contrasts `c.hm` and `c.mn`. `c.hm` is not significant (P-value 0.14) and `c.mn` is not significant either (P-value 0.20). This is the same significance as from Tukey, but note that the P-values for the non-significant tests are much lower than the corresponding ones from Tukey, once again because we have focused on just these comparisons, and not on any others. We decided ahead of time to test just these, and gave ourselves the best chance of finding significance that we could.

19.2 *Who studies the most outside class?*

A social scientist wanted to see how many hours students studied outside of class. She took a random sample of 75 students from three different majors: math, social science and English, and recorded the number of weekly outside-class study hours for each student. The data can be found at link. The social scientist had two research questions: whether math students study more outside of class than the other students, and whether English and social science students study different amounts of time outside class.

(a) Explain briefly why contrasts would be a better idea here than ANOVA with Tukey.

Solution

The researcher is not interested in all the comparisons between the groups, but rather in two specific ones (as detailed in the question). So we should focus our attention on those comparisons. This is what contrasts are for.

(b) Read in the data and make side-by-side boxplots of study hours by major. What do the boxplots suggest about the comparisons that the researcher wants to make?

Solution

Separated by one space. There appear to be some strange quotes in there, but we'll ignore those and see whether they cause any trouble:

```
my_url = "http://www.utsc.utoronto.ca/~butler/d29/studyhours.txt"
studyhours = read_delim(my_url, " ")
```

```
## Parsed with column specification:
## cols(
##   id = col_integer(),
##   major = col_character(),
##   hours = col_integer()
## )
```

```
studyhours
```

```
## # A tibble: 75 x 3
##       id major hours
##    <int> <chr> <int>
##  1     1 math     15
##  2     2 math     20
##  3     3 math     14
##  4     4 math     15
##  5     5 math     14
##  6     6 math     10
##  7     7 math     12
##  8     8 math      9
##  9     9 math     10
## 10    10 math     11
## # ... with 65 more rows
```

So far so good. 75 students, in tidy format.

```
ggplot(studyhours, aes(x = major, y = hours)) +
    geom_boxplot()
```

This suggests that the math students study more than the others, but the English and social science students study about the same amount.

I see some rather consistent right-skewness here (but no outliers). We have 25 observations in each group, a decent amount of data, so I'm not too worried, but a transformation could be a good idea, something like square root, perhaps. The spreads are not that unequal, but it *is* true that the Math students have both the largest median *and* the largest spread, and the social science students are the lowest on both.

Box-Cox also works on ANOVA-type data, since that is an `lm` kind of model:

```
boxcox(hours ~ major, data = studyhours)
```

This suggests that some kind of transformation would be a good idea (no transformation, power 1, is not supported by the data). My guess of 0.5 is not bad; that, or something a bit less like 0.25 (fourth

root) would be good. Even log is supportable. But we'll proceed without transformation in this question to give you some practice with contrasts. Despite appearances, I do (sometimes) like to make things not so complicated!

(c) Set up contrasts for each of the researcher's research questions, bearing in mind the levels of `major` and the order in which they come. (For the `Math` contrast, you want Math against the average of the other two.)

Solution

We need to know what the categories of `major` are, so that we can set up the contrasts correctly, so let's start with that:

```
studyhours %>% select(major) %>% summary()
```

```
##     major
##  Length:75
##  Class :character
##  Mode  :character
```

Oh. Yeah. We never turned `major` into a factor. We'd better do that, and then try again:

```
studyhours = studyhours %>% mutate(major = factor(major))
studyhours %>% select(major) %>% summary()
```

```
##      major
##  english:25
##  math   :25
##  socsci :25
```

In alphabetical order.

All right, contrasts. The English vs. social science one is easier, so we'll do that first. Just set it up with a 1 for one of them and a -1 for the other and 0 for math, in the right order according to what the levels are, viz:

```
c.eng.socsci = c(1, 0, -1)
```

Or `c(-1,0,1)` would be just as good.

The other one is a bit trickier, because you want math against the average of the others, like

$$\text{math} - (\text{English} + \text{socsci})/2.$$

This translates into contrast-ese like this, making sure to get Math in the middle where it belongs:

As I failed to do the first time.

```
c.math.others = c(-0.5, 1, -0.5)
```

The coefficients have to add up to zero, so I have to have two halves to balance my one 1. Or I can use two −1s and a 2.

(d) Verify that your two contrasts are orthogonal.

Solution

Multiply them together (elementwise, which is how R does it) and show that what you get adds up to zero:

```
sum(c.eng.socsci * c.math.others)
```

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

Zero. Orthogonal.

So we are safely in "familiar" territory, not in the here-be-dragons land of non-orthogonal contrasts.

On ancient maps, people didn't know what was in certain parts of the world, because no-one had ever explored them, so they wrote on the map *here be dragons*.

(e) Create a matrix of contrasts (using `cbind`) and let `lm` know that these are contrasts for `major`, in data frame `studyhours` (or whatever you called it).

Solution

So, like the example in class (where the `contrasts(model)` was admittedly rather confusing):

```
m = cbind(c.math.others, c.eng.socsci)
m
```

```
##      c.math.others c.eng.socsci
## [1,]          -0.5            1
## [2,]           1.0            0
## [3,]          -0.5           -1
```

```
contrasts(studyhours$major) = m
```

(f) Fit the model using `lm`. What do you conclude about your contrasts?

Solution

Pretend the ANOVA is a regression (though you can fit ANOVAs either way: we just used `aov` before because we were going to follow up with Tukey):

```
studyhours.1 = lm(hours ~ major, data = studyhours)
summary(studyhours.1)
```

```
##
## Call:
## lm(formula = hours ~ major, data = studyhours)
##
## Residuals:
##    Min     1Q Median     3Q    Max
##  -6.44  -2.48  -0.48   2.52  10.56
##
## Coefficients:
##                    Estimate Std. Error
## (Intercept)          7.3200     0.3980
## majorc.math.others   2.1200     0.5628
## majorc.eng.socsci    0.7800     0.4874
##                    t value Pr(>|t|)
## (Intercept)         18.392  < 2e-16 ***
## majorc.math.others   3.767 0.000335 ***
## majorc.eng.socsci    1.600 0.113936
## ---
## Signif. codes:
##   0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1  ' ' 1
##
## Residual standard error: 3.447 on 72 degrees of freedom
## Multiple R-squared:  0.1887, Adjusted R-squared:  0.1662
## F-statistic: 8.374 on 2 and 72 DF,  p-value: 0.0005375
```

There is a seriously significant difference between the math students and the others. There is not, however, any kind of difference between the mean study hours of the English and social science students.

(g) Are you surprised by the significance or non-significance you found in the previous part? Explain briefly.

Solution

Go back and compare with the boxplots. The math students studied noticeably more than the others, so it is not surprising that this difference came out significant. The median study times for the English and social science students were similar, so it is not at all surprising that this difference failed to be significant.

When you set up contrasts, *these* are the tests you are interested in, so there is no reason for following up with Tukey or anything else. But you have to be able to say ahead of time which contrasts you want to test. This is in contrast to Tukey, where you don't have to decide which comparisons interest you until right at the end.

In contrast. Get it? No? Well, never mind then.

Another question you might have had is "how many contrasts can I do"? To answer this, go back and look at the ANOVA table. If you

run an ANOVA using `lm`, you get the ANOVA table by passing the fitted model object into `anova` rather than `summary`:

anova is one of R's multi-purpose tools; what it does depends on what you feed it.

```
anova(studyhours.1)
```

```
## Analysis of Variance Table
##
## Response: hours
##           Df Sum Sq Mean Sq F value
## major      2 198.96   99.48  8.3737
## Residuals 72 855.36   11.88
##              Pr(>F)
## major     0.0005375 ***
## Residuals
## ---
## Signif. codes:
##   0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1  ' ' 1
```

Two degrees of freedom for `major`, so we can do 2 contrasts. (The example in class had four models of chainsaw, so 3 df for model, so 3 contrasts possible.) What if we don't use all those contrasts up? Try it and see:

```
m = cbind(c.math.others)
m
```

```
##      c.math.others
## [1,]          -0.5
## [2,]           1.0
## [3,]          -0.5
```

```
contrasts(studyhours$major) = m
studyhours.2 = lm(hours ~ major, data = studyhours)
summary(studyhours.2)
```

```
##
## Call:
## lm(formula = hours ~ major, data = studyhours)
##
## Residuals:
##    Min     1Q Median     3Q    Max
##  -6.44  -2.48  -0.48   2.52  10.56
##
## Coefficients:
##                    Estimate Std. Error
## (Intercept)          7.3200     0.3980
```

```
## majorc.math.others   2.1200     0.5628
## major               -1.1031     0.6893
##                    t value Pr(>|t|)
## (Intercept)         18.392  < 2e-16 ***
## majorc.math.others   3.767 0.000335 ***
## major               -1.600 0.113936
## ---
## Signif. codes:
##   0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1  ' ' 1
##
## Residual standard error: 3.447 on 72 degrees of freedom
## Multiple R-squared:  0.1887, Adjusted R-squared:  0.1662
## F-statistic: 8.374 on 2 and 72 DF,  p-value: 0.0005375
```

The two P-values are identical, but the second one is not now identified with any contrast (so we just ignore it). If we were only interested in how math students compared with the rest, we could test it this way.

Or, we could relabel the students as "math" and "other" and just do a two-sample *t*-test. To do this, we need to delve into `forcats` (which is a part of the `tidyverse`). It deals with categorical variables which are genuine factors, which we haven't had much cause to do so far. But now we do. What we want to do is to *recode* the levels of `major`, which is done by `fct_recode`:

```
studyhoursx = studyhours %>% mutate(mathrest = fct_recode(major,
    rest = "english", rest = "socsci"))
studyhoursx %>% count(mathrest)
```

```
## # A tibble: 2 x 2
##   mathrest     n
##   <fct>    <int>
## 1 rest        50
## 2 math        25
```

The way `fct_recode` works is that you first give a factor, here `major`, then you define a list of new levels in terms of the old ones. We want a student whose major is either English or Social Science to end up in the new not-math group that we'll call `rest`, so we need two lines inside `fct_recode`, one that says that the new `rest` group includes the old `english` majors, and another that says that `rest` also includes the old `socsci` majors. Any levels not mentioned (here `math`) are left unchanged.

I did this wrong the first time: I got the things before and after the `=` in `fct_recode` the wrong way around.

The last line is to check that the result makes sense. I previously had 25 students in each major. Now I have 25 students for whom `mathrest` is `math` (the math majors) and 50 (25 + 25) students for whom `mathrest` is `rest` (everyone else).

Now, the *t*-test.

```
t.test(hours ~ mathrest, data = studyhoursx, var.equal = T)
```

```
##
##  Two Sample t-test
##
## data:  hours by mathrest
## t = -3.7269, df = 73, p-value =
## 0.0003796
## alternative hypothesis: true difference in means is not equal to 0
## 95 percent confidence interval:
##  -4.880528 -1.479472
## sample estimates:
## mean in group rest mean in group math
##               6.26               9.44
```

This is very close to, but not the same as, the test with the contrasts, because we are not quite testing the same thing. Here, we are comparing the mean of the math students to the mean of everyone else, thrown together into one big bag, but with the contrasts, we are comparing the mean of the math students with the *average of the means* of the other two groups, treating them as different. (Not clear? Well, it's not that clear to me either.)

That business with the `var.equal=T`? This is doing a pooled *t*-test. I did that here because the ANOVA is assuming equal spreads within the groups (which is what a pooled *t*-test does) and I thought I should do my best to be consistent. A pooled *t*-test is really a two-group one-way ANOVA, or at least it is if it is done two-sided.

19.3 *Mental context*

It is believed that being in the same mental context for learning and for testing leads to better test scores. An experiment was carried out to test this. During the learning phase, subjects learned a list of 80 words in a room painted orange and decorated with posters, paintings and other paraphernalia. A memory test was given to all subjects immediately after they had learned the words, to give the impression that the experiment was over. (The results of this test were discarded.) One day later, subjects were unexpectedly re-tested under different experimental conditions and asked to write down all

This is a fancy word for *stuff*.

the words of the original list that they could remember. The re-test took place in five different conditions, which subjects were randomly allocated to one of:

- Same context: in the original orange-painted room.
- Different context: in a very different room, painted grey and located in a different part of campus.
- Imaginary context: in the grey room, but subjects are asked to remember the orange room in which they took the original test. To help them with this, the experimenter begins by asking them several questions about the orange room and the objects in it.
- Photographed context: in the grey room, but subjects are shown photographs of the orange room.
- Placebo context: in the grey room, with subjects first being asked to recall their living room.

In each case, the response variable was the number of words on the original list successfully recalled by each subject.

50 subjects in total completed the experiment, 10 under each of the 5 conditions.

The researchers had four research hypotheses to test with the data:

- Groups where the context matches (is the same, or is simulated by imagining or photograph) will perform better than groups with different or placebo contexts.
- The group with the same context will differ from the group with imaginary or photographed contexts.
- The imaginary-context group will differ from the photographed-context group.
- The different-context group will differ from the placebo-context group.

The research hypotheses are (as is usual) written as *alternative* hypotheses. We can rewrite them as null hypotheses, with some extra wording to facilitate converting them into contrasts, like this:

- The mean of the `same`, `imaginary` and `photograph` groups (group means) is equal to the mean of the `different` and `placebo` groups (group means).
- The mean of the `imaginary` and `photograph` groups is equal to the (mean of the) `same` group.

- The `imaginary` and the `photograph` groups will have the same mean.
- The `different` and the `placebo` groups will have the same mean.

The data are in link (the original researcher's name was Smith).

(a) Read in the data and verify that you have a column called `context` that is text and a column called `words` that is a (whole) number.

Solution

The usual thing — read in the data appropriately and look at the data frame you got:

```
my_url = "http://www.utsc.utoronto.ca/~butler/d29/smith.txt"
smith = read_delim(my_url, " ")
```

```
## Parsed with column specification:
## cols(
##   id = col_integer(),
##   context = col_character(),
##   words = col_integer()
## )
```

```
smith
```

```
## # A tibble: 50 x 3
##       id context words
##    <int> <chr>   <int>
##  1     1 same       25
##  2     2 same       26
##  3     3 same       17
##  4     4 same       15
##  5     5 same       14
##  6     6 same       17
##  7     7 same       14
##  8     8 same       20
##  9     9 same       11
## 10    10 same       21
## # ... with 40 more rows
```

`int` means "whole number", so we are good.

(b) Turn `context` into a `factor`, within the data frame. (We are going to be doing contrasts). Display how the data frame now looks.

Solution

Use `mutate` and assign everything back to what it was before:

```
smith = smith %>% mutate(context = factor(context))
smith
```

```
## # A tibble: 50 x 3
##       id context words
##    <int> <fct>   <int>
##  1     1 same       25
##  2     2 same       26
##  3     3 same       17
##  4     4 same       15
##  5     5 same       14
##  6     6 same       17
##  7     7 same       14
##  8     8 same       20
##  9     9 same       11
## 10    10 same       21
## # ... with 40 more rows
```

Note that `context` is now labelled as the factor that we made.

(c) What are the names of the five contexts in the data set (just display them), and what order are they in?

Solution

You have lots of choices now, since `context` is a factor (and can also be treated as text). The most obvious factor-based way is this:

```
levels(smith$context)
```

```
## [1] "different"  "imaginary"  "photograph"
## [4] "placebo"    "same"
```

or use `summary`:

```
smith %>% select(context) %>% summary()
```

```
##        context
##  different :10
##  imaginary :10
##  photograph:10
##  placebo   :10
##  same      :10
```

Or treat `context` as text, such as this way:

```
smith %>% count(context)
```

```
## # A tibble: 5 x 2
##   context        n
##   <fct>      <int>
## 1 different     10
## 2 imaginary     10
## 3 photograph    10
## 4 placebo       10
## 5 same          10
```

These all display the contexts in alphabetical order.

Or display the distinct values:

```
smith %>% distinct(context)
```

```
## # A tibble: 5 x 1
##   context
##   <fct>
## 1 same
## 2 different
## 3 imaginary
## 4 photograph
## 5 placebo
```

or, without using the `tidyverse`,

```
unique(smith$context)
```

```
## [1] same       different  imaginary
## [4] photograph placebo
## 5 Levels: different ... same
```

These two *don't* display things in alphabetical order, because they display things as they appear in the file. This is not the order we want, though the second way *does* display the `Levels` in the right order.

(d) Write each of the four research hypotheses (in the null-hypothesis versions) as R vectors that can be used to make contrasts. (This will mean getting the contexts in the right order. If you get stuck, do the last two first, since they're easiest. The first one is actually the hardest.)

Solution

All right, let's go from the bottom:

- Different and placebo have same means. These are 1st and 4th.

```
c4 = c(1, 0, 0, -1, 0)
```

- Imaginary and photograph have same means. 2nd and 3rd:

```
c3 = c(0, 1, -1, 0, 0)
```

- Imaginary and photograph together same as `same`. This time we have two means being compared with one, so we need to give the two means half weight. 2nd and 3rd against 5th:

```
c2 = c(0, 1/2, 1/2, 0, -1)
```

- (Same and imaginary and photograph) vs. (different or placebo): 2nd, 3rd and 5th with weight 1/3 (three of them) against 1st and 4th with weight 1/2 (2 of them):

```
c1 = c(-1/2, 1/3, 1/3, -1/2, 1/3)
```

Variations: you can switch the sign on everything within a contrast (since it doesn't matter which is plus and which is minus). You can also multiply through by anything to get rid of the fractions, for example these:

```
c2 = c(0, 1, 1, 0, -2)
c1 = c(-3, 2, 2, -3, 2)
```

All that matters is that the coefficients add up to zero, and that they are the right size and sign relative to each other.

(e) Pick two of your contrasts (doesn't matter which two) and demonstrate that they are orthogonal.

Solution

Multiply your chosen contrasts together elementwise, and show that the results add to zero, eg. by showing the whole thing:

```
c1 * c2
```

```
## [1]  0.0000000  0.1666667  0.1666667
## [4]  0.0000000 -0.3333333
```

which adds to zero because it is 2/6 minus 1/3, or by explicitly summing the elementwise product:

```
sum(c1 * c3)
```

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

Any pair is good.

(f) Collect your contrasts together into a matrix, and tell `lm` that these are the contrasts for `context`.

Solution

This:

```
m = cbind(c1, c2, c3, c4)
contrasts(smith$context) = m
```

Slipping off into another aside, you might have been wondering whether there was a way to test *all* the contrasts for orthogonality at once. There is, and it depends on the rules for matrix multiplication. We want to test all the *columns* of `m` for orthogonality, but matrix multiplication works by combining a *row* with a column. No matter, *transposing* a matrix interchanges rows and columns, so that in math, we want to look at the matrix $M^T M$. In R, `%*%` means "matrix multiply". Thus,

In R, percents around something mean that it is a special version of that something. Hence the notation for matrix-multiply and the pipe symbol. A regular * when used for multiplying matrices in R will multiply them element by element.

```
t(m) %*% m
```

```
##           c1  c2 c3 c4
## c1 0.8333333 0.0  0  0
## c2 0.0000000 1.5  0  0
## c3 0.0000000 0.0  2  0
## c4 0.0000000 0.0  0  2
```

This matrix-multiplies the transpose of `m` by `m` itself. There are numbers down the top-left to bottom-right diagonal, but don't worry about these, since a contrast doesn't have to be orthogonal with itself. The thing to note is that *all the other elements* of this matrix are *zero*: that means that each of the four contrasts is orthogonal to each of the other three.

(g) Fit a model with `lm`, and display the results.

Solution

We're past the hard part:

```
smith.1 = lm(words ~ context, data = smith)
summary(smith.1)
```

```
##
## Call:
## lm(formula = words ~ context, data = smith)
##
## Residuals:
##    Min     1Q Median     3Q    Max
##   -9.0   -4.0   -1.5    4.6   11.6
##
## Coefficients:
```

```
##             Estimate Std. Error t value
## (Intercept)  14.8000     0.8129  18.207
## contextc1     8.6000     1.9912   4.319
## contextc2    -0.3333     1.4841  -0.225
## contextc3    -0.1000     1.2853  -0.078
## contextc4     0.5000     1.2853   0.389
##             Pr(>|t|)
## (Intercept)  < 2e-16 ***
## contextc1   8.52e-05 ***
## contextc2      0.823
## contextc3      0.938
## contextc4      0.699
## ---
## Signif. codes:
##   0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1  ' ' 1
##
## Residual standard error: 5.748 on 45 degrees of freedom
## Multiple R-squared:  0.2954, Adjusted R-squared:  0.2327
## F-statistic: 4.715 on 4 and 45 DF,  p-value: 0.002902
```

(h) For each of the original research hypotheses, what do you infer about them?

Solution

Even though I generated the contrasts backwards, I maintained the numbering so that they refer to the original numbered hypotheses. You might have named them something more mnemonic; that works too, and is quite possibly better. Anyway:

- Matching context better than non-matching context: strongly significant (P-value 0.000085). (Strictly, this is a two-sided test, but the way I constructed the contrast, this should be significantly *positive* if the research hypothesis is true, and it is.)
- Same context *not* different from imaginary/photographed context (P-value 0.823)
- Imaginary context *not* different from photographed context (P-value 0.938).
- Different context *not* different from placebo context (P-value 0.699).

I wanted you to match up the research hypotheses with the P-values, as above, and state a conclusion about each one. If you do that, I am happy. Taking this back to the original research, I think the first hypothesis was the most immediately important of the four: we were able to show that having (or faking up) the original room

helped with recall. After that, it didn't matter how it was done: being in the original room was not different from imagining the original room (by thinking about it or seeing a photo of it). Failing to recall the original room was equally bad, whether the subjects were in a different room and not asked to think about the original orange room, or whether they were literally asked to think about another room entirely. Contrasts enabled us to tackle this problem and gain a very clear conclusion: recalling the original orange room is helpful, but it doesn't matter how you do it. How big of a difference does thinking about the orange room make? You can use the 2 SE thing to get a 95% confidence interval: the difference in the (mean of the) recall-orange-room means and the (mean of the) don't-recall means is between about 4.6 and 12.6 words (out of 80). I leave it for you to decide whether that is practically important as well as being statistically significant.

19.4 Trying on shirts

Does it make a difference if you see somebody else trying on a shirt before you do, and if so, does it matter what kind of person it is?

An experiment was carried out in a university bookstore, with students who wanted to try on a shirt serving as (unsuspecting) experimental subjects. When a student wanted to try on a shirt, the sales associate told them that there was only one shirt left, and it was being tried on by an "other customer". This "other customer" was actually a "confederate" of the experimenter (that means, they were pretending to be a real customer but were actually part of the experiment). The "other customer" was always female: either an attractive well-dressed model, or an average-looking student wearing jeans. The "other customer" would come out of the dressing room and hand the shirt to the sales associate, who would give it to the student who wanted to try it on. When the student had tried on the shirt, they were asked to rate it on a 7-point scale (on five different dimensions, and those five scores were then averaged). Does a student's evaluation depend on whether the student was male or female, and whether the "other customer" was a model or a student? There was also a control group, where the student was handed the shirt directly by the sales associate, without the "other customer" being involved at all.

There were thus five treatments: male students who saw a model, male students who saw a student, female students who saw a model, female students who saw a student, and the control group. There were 25 students in each treatment group.

The data from this experiment can be found at link.

(a) Read in and display the data. How many observations do you have, and is that what you should have?

Solution
`read_csv`:

```
my_url = "http://www.utsc.utoronto.ca/~butler/d29/shirts.csv"
shirts = read_csv(my_url)
```

```
## Parsed with column specification:
## cols(
##   treatment = col_character(),
##   score = col_double()
## )
```

```
shirts
```

```
## # A tibble: 125 x 2
##    treatment         score
##    <chr>             <dbl>
##  1 male_seeing_model   6.8
##  2 male_seeing_model   4.7
##  3 male_seeing_model   5.1
##  4 male_seeing_model   4.7
##  5 male_seeing_model   5.5
##  6 male_seeing_model   6
##  7 male_seeing_model   4.4
##  8 male_seeing_model   6
##  9 male_seeing_model   5.3
## 10 male_seeing_model   6.3
## # ... with 115 more rows
```

There were 5 treatment groups with 25 students in each, so there should be, and are, $5 \times 25 = 125$ observations.

(b) Turn `treatment` into a `factor` in your data frame. (You can use the same name as the text `treatment` that you read in from the file.)

Solution
Thus:

```
shirts = shirts %>% mutate(treatment = factor(treatment))
```

and for checking:

```
shirts
```

```
## # A tibble: 125 x 2
##    treatment         score
##    <fct>             <dbl>
##  1 male_seeing_model   6.8
##  2 male_seeing_model   4.7
##  3 male_seeing_model   5.1
##  4 male_seeing_model   4.7
##  5 male_seeing_model   5.5
##  6 male_seeing_model   6
##  7 male_seeing_model   4.4
##  8 male_seeing_model   6
##  9 male_seeing_model   5.3
## 10 male_seeing_model   6.3
## # ... with 115 more rows
```

see that `treatment` is now definitely a factor rather than a categorical variable represented as a piece of text.

(c) List the treatments in the order that they are in within your `factor`. (There are lots of ways to do this; any one of them except for `distinct` is good.)

Solution
This is the most direct:

```
levels(shirts$treatment)
```

```
## [1] "control"
## [2] "female_seeing_model"
## [3] "female_seeing_student"
## [4] "male_seeing_model"
## [5] "male_seeing_student"
```

or you can do one of the `tidyverse`-style ways, such as

```
shirts %>% count(treatment)
```

```
## # A tibble: 5 x 2
##   treatment                 n
##   <fct>                 <int>
## 1 control                  25
## 2 female_seeing_model      25
## 3 female_seeing_student    25
## 4 male_seeing_model        25
## 5 male_seeing_student      25
```

or

```
shirts %>% distinct(treatment)
```

```
## # A tibble: 5 x 1
##   treatment
##   <fct>
## 1 male_seeing_model
## 2 male_seeing_student
## 3 female_seeing_model
## 4 female_seeing_student
## 5 control
```

except that `distinct` shows you the values in the order that they appear in the data file, which is *not* the order that they appear in the factor. So that's why I asked you not to do it that way.

Alphabetical order, is the moral of the story.

Extra: you can make them appear in the order they appear in the data, if you want to (and if you know what you are doing):

```
d <- shirts %>% mutate(trt2 = fct_inorder(treatment))
levels(d$trt2)
```

```
## [1] "male_seeing_model"
## [2] "male_seeing_student"
## [3] "female_seeing_model"
## [4] "female_seeing_student"
## [5] "control"
```

`fct_inorder` *creates* a factor whose levels are in the order that they appeared in the data. If this is what you want, you can go ahead and do this, and use this order when you create your contrasts. It will work, and it will be good even though your contrasts will look different from mine.

(d)* Obtain a table of mean evaluation scores for each treatment group.

Solution

We will use this later when assessing the significance of the contrasts. It's the usual group-by and summarize:

```
shirts %>% group_by(treatment) %>% summarize(m = mean(score))
```

```
## # A tibble: 5 x 2
##   treatment                 m
##   <fct>                 <dbl>
## 1 control                4.03
## 2 female_seeing_model    4.07
## 3 female_seeing_student  3.36
## 4 male_seeing_model      5.42
## 5 male_seeing_student    3.25
```

Extra: how does this come out if you have your factor levels in a non-alphabetical order?

```
d %>% group_by(trt2) %>% summarize(m = mean(score))
```

```
## # A tibble: 5 x 2
##   trt2                     m
##   <fct>                <dbl>
## 1 male_seeing_model     5.42
## 2 male_seeing_student   3.25
## 3 female_seeing_model   4.07
## 4 female_seeing_student 3.36
## 5 control               4.03
```

Ah, it respects the order of the levels of the factor: that is to say, the treatment groups appear here in the order that they were in the data, because the factor `trt2` was created to make that happen.

`forcats` has other tools to re-order the levels of a factor, lump factor levels together to make a category "other", and so on. These also work if you start with a categorical variable as text: it *creates* a factor with those properties.

(e) The experimenters wanted to compare four specific things in their analysis:

- evaluation scores between male students who saw a (female) model and male students who saw a (female) student
- evaluation scores between female students who saw a (female) model and female students who saw a (female) student
- evaluation scores for male and for female students (overall)
- evaluation scores for the (average of the) four genuine treatments and for the control group

Create contrasts, with suitable names, using vectors with appropriate values.

Solution

These, in that order, are comparisons of treatments 4 and 5:

```
c_mms = c(0, 0, 0, 1, -1)
```

treatments 2 and 3:

```
c_fms = c(0, 1, -1, 0, 0)
```

(in these two the 1 and -1 can also be the other way around)

the average of 2 and 3 vs. the average of 4 and 5:

```
c_mf = c(0, 0.5, 0.5, -0.5, -0.5)
```

or you can use 1 and -1 instead of the 0.5s, or you can switch the signs around. I like the 0.5 values to remind me it's an average of two means.

Finally

```
c_tc = c(1, -0.25, -0.25, -0.25, -0.25)
```

or multiply through by 4 to get rid of the fractions, or have the negative sign on the first one and positive signs on the others.

I tried to give the contrasts mnemonic but short names, so that I would remember which was which.

(f) Pick two of your contrasts (doesn't matter which two) and demonstrate that they are orthogonal.

Solution

Multiply them together (elementwise, which is what $*$ does) and add them up, showing that you get zero, for example:

```
sum(c_mf * c_tc)
```

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

Check (mops brow).

(g) Collect all your contrasts together into a matrix and declare that they are contrasts for `treatment` within your data frame (whatever you called it).

Solution

I called my data frame `shirts`, so I need to do this:

```
m = cbind(c_mms, c_fms, c_mf, c_tc)
contrasts(shirts$treatment) = m
```

There's no output here; we'll see in a moment whether it worked.

Extra: once you've created the matrix `m`, it gives you a second way to test the contrasts for orthogonality, all at once:

```
t(m) %*% m
```

```
##       c_mms c_fms c_mf c_tc
## c_mms     2     0    0 0.00
## c_fms     0     2    0 0.00
## c_mf      0     0    1 0.00
## c_tc      0     0    0 1.25
```

This matrix-multiplies the transpose of `m` by `m`, so it's testing each column of `m` to see whether it's orthogonal to each other column. The columns don't have to be orthogonal to themselves, hence the non-zeros down the diagonal, but all the off-diagonal entries are zero. Hence each contrast is orthogonal to each other contrast.

Or even:

```
z = t(m) %*% m
all(z[row(z) != col(z)] == 0)
```

```
## [1] TRUE
```

That says (a little breathlessly) that it is true that all the elements of $M^T M$ that are off the diagonal are zero.

The thing inside the square brackets says only to look at the elements of $M^T M$ whose row number and whose column number are different; it is perhaps easier to reason that elements of a matrix whose row number and column number are the *same* are *on* the diagonal, for example the element in row 2, column 2.

(h) Predict evaluation score from treatment *as a regression*, and display the results.

Solution

Once you have everything set up, it's just a matter of going through the process:

```
score.1 = lm(score ~ treatment, data = shirts)
summary(score.1)
```

```
##
## Call:
## lm(formula = score ~ treatment, data = shirts)
##
## Residuals:
##    Min     1Q Median     3Q    Max
## -3.860 -0.760  0.032  0.840  3.640
##
## Coefficients:
##                Estimate Std. Error t value
## (Intercept)      4.0256     0.1231  32.705
## treatmentc_mms   1.0820     0.1946   5.560
## treatmentc_fms   0.3540     0.1946   1.819
## treatmentc_mf   -0.6200     0.2752  -2.253
## treatmentc_tc    0.0064     0.2462   0.026
##                Pr(>|t|)
## (Intercept)     < 2e-16 ***
## treatmentc_mms 1.66e-07 ***
## treatmentc_fms   0.0714 .
## treatmentc_mf    0.0261 *
## treatmentc_tc    0.9793
```

```
## ---
## Signif. codes:
##   0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1  ' ' 1
##
## Residual standard error: 1.376 on 120 degrees of freedom
## Multiple R-squared:  0.2467, Adjusted R-squared:  0.2216
## F-statistic: 9.823 on 4 and 120 DF,  p-value: 6.576e-07
```

Note how the contrasts have appeared as "slopes" in our regression. (If this didn't happen for you, go back and check what you did. Probably something gave you an error before, in that case.)

Your slopes can differ in terms of sign and possibly value from mine, but the t-statistics and P-values should be the same as mine.

(i) For each of your contrasts, assess whether or not it is significant, and explain briefly what that means in the context of the data. If a contrast is significant, use your answer to part (here) to help in your interpretation.

Solution

I'll take my four contrasts as they appear in the output from `lm`:

- Comparing the male students who saw the model with those who saw the student: this is strongly significant, and the table of means shows that the males who saw the model rated the shirt significantly *higher* than those who saw the "average-looking" female student.
- Comparing the female students who saw the model with those who saw the student: this difference is not quite significant, so we conclude that it didn't matter who the female students saw coming out of the dressing room.
- There is a significant difference between the average evaluation score that males gave to the shirt and that females did. The table of means reveals that the male average was higher, mostly because males *really* liked the shirt when they thought the model had been wearing it!
- There is no significant difference overall between treatment and control groups in terms of average score. (This is mostly because the treatment groups differ among themselves, with the scores for students who saw the model being higher and those for students who saw the female student being lower than for the control group.)

It is enough, on the non-significant ones, to say what it is that the contrast is testing, and then to note that it is not significant. On the

significant ones, I want you to say what is higher than what. You can also get these conclusions from the `lm` output, but it requires a bit more care:

- I set up `c_mms` with the score for the students who had seen the model as plus and those who saw the student as minus. My slope here is positive, so the score for male students who saw the model is higher.
- My slope for `c_fms` is also positive, and I set this one up the same way with model as positive and student as negative, so the female students rated the shirt higher (but not significantly higher) when they saw the model.
- My `c_mf` slope is significantly negative. I set up this contrast with females as plus and males as minus, so this means males gave a significantly *higher* score on average (for reasons discussed above).
- The last slope is very close to 0, consistent with there being no difference (on average) between treatment and control.

Perhaps the story to get from this analysis is that male students are *such* suckers!

19.5 *Productivity and research-and-development*

An economist compiled data on productivity improvements for a sample of companies that produce electronic equipment. The companies were classified according to their expenditures on research and development in the last three years (classified as low, moderate, high). Productivity improvement is measured on a 0–100 scale, with most values being between 0 and 10. The economist also thought that last year's productivity improvement would be relevant, and so recorded this as well.

The data set can be found at link.

Feel free to use "R&D" as an abbreviation for "research and development" in your answers below.

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

Solution

The data values are separated by one space, so let's use `read_delim` and display whatever displays (probably the first ten lines):

```
my_url = "http://www.utsc.utoronto.ca/~butler/d29/productivity.txt"
productivity = read_delim(my_url, " ")
```

```
## Parsed with column specification:
## cols(
##   expenditure = col_character(),
##   improvement = col_double(),
##   last = col_double()
## )
```

```
productivity
```

```
## # A tibble: 27 x 3
##    expenditure improvement  last
##    <chr>             <dbl> <dbl>
##  1 low                 7.6   8.2
##  2 low                 8.2   7.9
##  3 low                 6.8   7
##  4 low                 5.8   5.7
##  5 low                 6.9   7.2
##  6 low                 6.6   7
##  7 low                 6.3   6.5
##  8 low                 7.7   7.9
##  9 low                 6     6.3
## 10 moderate            6.7   8.8
## # ... with 17 more rows
```

There is a column classifying expenditure along with numerical values for productivity improvement for this year and last year (labelled `improvement` and `last`), so it looks as if we are good.

You are free to give the data frame a shorter name to make it easier to keep typing it!

Extra: those expenditure levels (that are currently text) will get sorted into alphabetical order unless we stop them. They are actually in the right order in the data file, so maybe we should turn them into a factor (with the levels in the right order) now:

```
productivity <- productivity %>% mutate(expenditure = fct_inorder(expenditure))
```

If you don't do this, your answers below will be different from mine. Probably not in any kind of consequential way, but different nonetheless.

(b) Plot productivity improvement for this year against last year, identifying the points by the level of research-and-development expenditure. On your plot, add regression lines for each expenditure group.

Solution

Two quantitative and one categorical variable, so plot the categorical variable using colour (or `shape` etc., if you know about that, but colour is the most obvious thing):

```
ggplot(productivity, aes(x = last, y = improvement,
    colour = expenditure)) + geom_point() + geom_smooth(method = "lm",
    se = F)
```

The `se=F` is optional; no problem if you don't include it. (If you miss it out, you'll get those CI-of-mean-response grey envelopes around each line.)

(c)* Look at your graph of the previous part. Do you think that (i) knowing last year's productivity improvement tells you something about this year's, (ii) the level of expenditure on research-and-development has an impact on this year's productivity improvement, (iii) there is an interaction effect between R&D expenditure and last year's productivity increase on this year's productivity increase? In each case, explain briefly.

Solution

Taking the three things in turn: (i) knowing about last year's productivity increase definitely helps, because the three trends definitely go up (extra: and are pretty much linear). (ii) knowing about the level of expenditure helps because the coloured trends are in different places (on mine, green on the left and red on the right). (iii) I am guessing there is no interaction because the three lines look more or less parallel (a judgement call: if you think that there *will* be an interaction because you think the lines are definitely not "approximately parallel" and therefore there is an interaction, that's OK too).

(d) Fit an analysis of covariance model without interaction. Run the results through `drop1` with `test="F"` (the `F` must be in quotes). What do you conclude?

Solution

This looks exactly like a regression with a categorical variable, and is. Just the two main effects, thus:

```
improvement.1 = lm(improvement ~ last + expenditure,
    data = productivity)
drop1(improvement.1, test = "F")
```

```
## Single term deletions
##
## Model:
## improvement ~ last + expenditure
##             Df Sum of Sq     RSS     AIC
## <none>                    1.3175 -73.542
## last         1   14.0447 15.3622  -9.226
## expenditure  2    4.1958  5.5134 -38.894
##             F value    Pr(>F)
## <none>
## last        245.176 9.274e-14 ***
## expenditure  36.623 7.095e-08 ***
## ---
## Signif. codes:
##   0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1  ' ' 1
```

Both the P-values are small, so there is strong evidence of an effect on this year's productivity increase of last year's productivity increase and the level of R&D expenditure.

(e) Now look at the `summary` of your analysis of covariance model. Explain briefly how each of the *last three* numbers in the `Estimate` column are consistent with your graph.

Solution

To begin:

```
summary(improvement.1)
```

```
##
## Call:
## lm(formula = improvement ~ last + expenditure, data = productivity)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -0.52812 -0.16385 -0.00046  0.08379  0.45730
##
## Coefficients:
```

```
##                      Estimate Std. Error
## (Intercept)          -1.00804    0.50991
## last                  1.11417    0.07116
## expendituremoderate -1.83316    0.22372
## expenditurehigh      -3.14338    0.37115
##                      t value Pr(>|t|)
## (Intercept)           -1.977   0.0602 .
## last                  15.658 9.27e-14 ***
## expendituremoderate  -8.194 2.84e-08 ***
## expenditurehigh      -8.469 1.59e-08 ***
## ---
## Signif. codes:
##   0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1  ' ' 1
##
## Residual standard error: 0.2393 on 23 degrees of freedom
## Multiple R-squared:  0.9629, Adjusted R-squared:  0.958
## F-statistic: 198.8 on 3 and 23 DF,  p-value: < 2.2e-16
```

The last three numbers in the Estimate column are slopes for the quantitative variable `last` and changes in intercept for the levels of the categorical variable `expenditure`. Specifically:

- the slope for `last` is positive, so if last year's productivity increase is higher, so will this year's. This matches the upward trends within each `expenditure` group on the graph.
- `expenditure` is categorical, so everything is measured relative to the baseline. Mine is `low`, but yours is probably `high`.
- My two estimates for `expenditure` `high` and `moderate` are both negative, so when expenditure on R&D is high or moderate, the productivity increase will be lower this year (for the same productivity increase last year). Going back to the graph, if you extend the lines for `moderate` and `high` to a middling `last` value of something like 10, the productivity increase this year will be higher when `expenditure` is low or moderate then when it is high, which is consistent with those slopes both being positive. (You don't need to go to this detail, but you should say something convincing from the graph about how this year's productivity increase is *higher* if R&D expenditure is low or moderate compared to high.)

Extra: the graph also indicates that the higher the `expenditure` on R&D is, the more likely last year's productivity improvement will be higher also. So in practice we are not likely to be comparing actual companies with different `expenditure` but the same `last`. Nonetheless, what I said is what those positive coefficients for `expenditure`

actually mean. Be careful not to say that `low` is lowest because it's on the left; it's whether it's higher or lower than the others *at the same value of* `last` that matters.

Extra extra: I asked you to do the test from `drop1` because there are three levels of `expenditure` (rather than just two), and the `summary` output only gives you a comparison with the baseline level, rather than comparing all three levels. I prefer to do the right test (using `drop1`) first, and then use `summary` to interpret what I have, or at least the parts of it that are significant.

Extra extra extra: you would (or at least I would) expect a larger productivity increase to go with a *larger* expenditure on R&D, but that's not how it worked out. This is one of those cases where all else isn't really equal.

(f) Add an interaction between `last` and `expenditure` to your analysis of covariance model. Test it for significance using `drop1`. In the light of what you have seen, does this surprise you? Explain briefly why you are or are not surprised.

Solution

I like `update` for this (writing out the whole model is an alternative):

```
improvement.2 = update(improvement.1, . ~ . +
    last:expenditure)
drop1(improvement.2, test = "F")
```

```
## Single term deletions
##
## Model:
## improvement ~ last + expenditure + last:expenditure
##                  Df Sum of Sq     RSS
## <none>                        0.95718
## last:expenditure  2   0.36035 1.31753
##                      AIC F value  Pr(>F)
## <none>           -78.169
## last:expenditure -73.542   3.953 0.03491 *
## ---
## Signif. codes:
##   0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1  ' ' 1
```

Yes, I too forgot the `test="F"` the first time.

The interaction term is actually significant, at least at $\alpha = 0.05$. This is a surprise to me, because I thought those lines on the graph were pretty close to parallel, so I wasn't expecting to see a significant interaction. (What I want from you here is to look back at your

answer to (iii) in part (here), and to say how this small P-value is consistent or inconsistent with whatever you said there. If you said that the lines weren't parallel enough, then this significant interaction should *not* be a surprise to you.)

That's all I'm after. I don't need you to speculate on *why* the test came out significant. I will, in a moment, but you don't need to.

Extra: I didn't expect the interaction to come out significant, but the P-value is not nearly so small as the ones we had before. I guess the explanation for this is that the data cluster pretty tightly about the lines in the graph, so that even a small difference in slopes can be signficant. You might argue that my green line is not *quite* as steep as the others, but the difference does look very small.

Let's take a look at the `summary` for my interaction model:

```
summary(improvement.2)
```

```
##
## Call:
## lm(formula = improvement ~ last + expenditure + last:expenditure,
##     data = productivity)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -0.32417 -0.14885 -0.02465  0.13739  0.55556
##
## Coefficients:
##                          Estimate Std. Error
## (Intercept)               0.27827    0.64967
## last                      0.93243    0.09124
## expendituremoderate      -4.50268    1.25959
## expenditurehigh          -7.14795    1.91223
## last:expendituremoderate  0.32217    0.14243
## last:expenditurehigh      0.40858    0.17549
##                          t value Pr(>|t|)
## (Intercept)                0.428  0.67278
## last                      10.220 1.32e-09
## expendituremoderate       -3.575  0.00179
## expenditurehigh           -3.738  0.00121
## last:expendituremoderate   2.262  0.03444
## last:expenditurehigh       2.328  0.02997
##
## (Intercept)
## last                     ***
## expendituremoderate      **
## expenditurehigh          **
```

```
## last:expenditurmoderate *
## last:expenditurehigh     *
## ---
## Signif. codes:
##   0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1  ' ' 1
##
## Residual standard error: 0.2135 on 21 degrees of freedom
## Multiple R-squared:  0.973,  Adjusted R-squared:  0.9666
## F-statistic: 151.5 on 5 and 21 DF,  p-value: 9.763e-16
```

The last two Estimates, the ones beginning with `last:`, came from the interaction. Once again `low` is my baseline. These say how the *slopes* for `last` for the other groups differ from the slopes for `last` for `low` expenditure. (Yes, this is confusing. Read it again until you get it.)

The biggest-size difference in slopes, about 0.40, is between low expenditure and high expenditure. This says that the line for high expenditure is this much more steep than the line for low expenditure. That's where the non-parallelism is on the graph, such as it is. A small but significant difference in slopes. You can tell that the effect is small by looking in the Estimate column; those changes in slopes, 0.40 and 0.32, are the smallest things in size out of everything in that column.

Having seen this, you now realize that I did this question wrong (or, at least, I led you through it wrong). The right way to do this would have been to fit the interaction model *first*, see that it is significant, and then done some predictions to assess the effects of things:

```
lasts = c(7, 9, 11)
expenditures = c("low", "moderate", "high")
new = crossing(last = lasts, expenditure = expenditures)
new
```

```
## # A tibble: 9 x 2
##    last expenditure
##   <dbl> <chr>
## 1     7 high
## 2     7 low
## 3     7 moderate
## 4     9 high
## 5     9 low
## 6     9 moderate
## 7    11 high
## 8    11 low
## 9    11 moderate
```

(oh, how I wish I'd given those variables shorter names), and then:

The expenditure levels have come out in alphabetical order again.

```
p = predict(improvement.2, new)
cbind(new, p)
```

```
##   last expenditure         p
## 1    7        high  2.517338
## 2    7         low  6.805256
## 3    7    moderate  4.557743
## 4    9        high  5.199343
## 5    9         low  8.670110
## 6    9    moderate  7.066929
## 7   11        high  7.881348
## 8   11         low 10.534963
## 9   11    moderate  9.576115
```

The effect is more or less as we described it before: as `last` goes up (for fixed `expenditure`), the predicted productivity improvements for this year go up, and as `expenditure` level goes up, the predictions go *down*. But the rates at which they go up or down are different, which is the significant interaction coming into play. Having said that, they are not very different, so I cheated and pretended the interaction was not significant (or that I was using $\alpha = 0.01$), so that you would have something easier to interpret. Qualitatively, the story is the same either way, because the sizes of the interaction terms are small compared to the others. So, even though I cheated, you ended up with more or less the same conclusions doing it the way I asked you to do it, or doing it the way I just did it.

19.6 *Treating leprosy*

Two drugs are being tested in the treatment of leprosy. These are labelled A and D. There is also a control drug, labelled F. The response variable is a post-treatment score of leprosy bacilli (measured at six different sites on each patient). A lower score is better.

Thus far, we have a standard one-way analysis of variance. But the researchers wanted greater precision in assessing the effects (if any) of the drugs, so they also measured a pre-treatment score of leprosy bacilli. The data are in the file link. The pre-treatment and post-treatment scores are labelled `pre` and `post` respectively.

(a) Read in the data and check that you have apparently the right thing.

Solution

Take a look at the data file. The values have multiple spaces between them, but they are aligned with each other and the column headings, so `read_table` is the thing:

```
my_url = "http://www.utsc.utoronto.ca/~butler/d29/leprosy.txt"
lepro = read_table(my_url)
```

```
## Parsed with column specification:
## cols(
##   drug = col_character(),
##   pre = col_integer(),
##   post = col_integer()
## )
```

```
lepro
```

```
## # A tibble: 30 x 3
##    drug    pre  post
##    <chr> <int> <int>
##  1 A        11     6
##  2 A         6     4
##  3 D         6     0
##  4 D         8     4
##  5 F        16    13
##  6 F        16    12
##  7 A         8     0
##  8 A        10    13
##  9 D         6     2
## 10 D        19    14
## # ... with 20 more rows
```

Call it what you like.

That looks good, with variables of the right names.

(b)* Make a scatterplot of post-treatment score against pre-treatment score, with the points for each drug drawn in a different colour.

Solution

This is the kind of thing that `ggplot` does without batting an eyelid:

```
ggplot(lepro, aes(x = pre, y = post, colour = drug)) +
    geom_point()
```

(c) Does it appear that including the pre-treatment score was a good idea? Explain briefly.

Solution

The overall trend on the scatterplot is that a higher `pre` tends to go with a higher `post`, regardless of drug, so including this information appears to be informative. I personally suspect that there's some fan-out happening on the pre-post relationship, but I'm not planning to make you explore that.

(d) What about this dataset suggests that analysis of covariance is a method worth trying?

Solution

The key is a mixture of categorical and quantitative explanatory variables. Here we have a categorical variable `drug` and a quantitative one `pre`. If we had only one type of explanatory variable, we could do a regression or an ANOVA as appropriate. But we don't. In some ways, it's not worth making a fuss about the distinction, because regressions and ANOVAs are all linear models anyway. But you may see the term "analysis of covariance", so it's worth your while to know what it's about.

(e) Fit an analysis of covariance model to predict post-treatment score. Include an interaction between your explanatory variables. (You don't need to look at the output from the model.)

Solution

This is what you'd guess. `lm` handles the interaction properly, even though `pre` is a quantitative variable.

```
lepro.1 = lm(post ~ pre * drug, data = lepro)
```

I wanted to take a look, so I did:

```
summary(lepro.1)
```

```
##
## Call:
## lm(formula = post ~ pre * drug, data = lepro)
##
## Residuals:
##    Min     1Q Median     3Q    Max
## -6.225 -2.437 -0.586  1.126  8.775
##
## Coefficients:
##             Estimate Std. Error t value
## (Intercept)  -1.6306     2.9455  -0.554
## pre           0.7452     0.2849   2.616
## drugD        -2.9549     4.1246  -0.716
## drugF        -1.4780     5.4678  -0.270
## pre:drugD     0.3233     0.3846   0.841
## pre:drugF     0.4492     0.4458   1.008
##             Pr(>|t|)
## (Intercept)   0.5850
## pre           0.0152 *
## drugD         0.4806
## drugF         0.7892
## pre:drugD     0.4089
## pre:drugF     0.3236
## ---
## Signif. codes:
##   0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1  ' ' 1
##
## Residual standard error: 4.07 on 24 degrees of freedom
## Multiple R-squared:  0.6915, Adjusted R-squared:  0.6272
## F-statistic: 10.76 on 5 and 24 DF,  p-value: 1.63e-05
```

For testing the interaction, there are *two* slope coefficients that should be zero if there is no interaction. So we have to test this with `drop1`, which is next.

(f) Pass your fitted model of the last part into `drop1`. Is the interaction term significant?

Solution
Just this:

```
drop1(lepro.1, test = "F")
```

```
## Single term deletions
##
## Model:
## post ~ pre * drug
##          Df Sum of Sq    RSS    AIC F value
## <none>                397.56 89.524
## pre:drug  2    19.645 417.20 86.971   0.593
##          Pr(>F)
## <none>
## pre:drug 0.5606
```

There is only a test for the interaction term because you can't take out the main effects until you've taken out the interaction.

The P-value for the interaction is very large (0.5606) so it is nowhere near significant. We can drop the interaction.

(g) Fit a model without the interaction. Is this a sensible thing to do (in addition, that is, to the fact that I just asked you to do it)?

Solution

Change the * to a +:

```
lepro.2 = lm(post ~ pre + drug, data = lepro)
```

Or use `update` (not much in it, here):

```
lepro.2a = update(lepro.1, . ~ . - pre:drug)
```

We just said that the interaction could come out, since it wasn't significant, so this is exactly the model that we should be fitting.

(h) Take a look at the `summary` of your preferred model. Is there a significant effect of pre-treatment score? Describe the effects of the different drugs on the post-treatment score. (Don't do any tests for `drug`.) Does your comparison of drugs make sense?

Solution

Mine was the no-interaction model `lepro.2`:

```
summary(lepro.2)
```

```
##
## Call:
## lm(formula = post ~ pre + drug, data = lepro)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -6.4115 -2.3891 -0.5711  1.7237  8.5885
```

```
##
## Coefficients:
##             Estimate Std. Error t value
## (Intercept)  -3.8808     1.9862  -1.954
## pre           0.9872     0.1645   6.001
## drugD         0.1090     1.7951   0.061
## drugF         3.4461     1.8868   1.826
##             Pr(>|t|)
## (Intercept)   0.0616 .
## pre         2.45e-06 ***
## drugD         0.9521
## drugF         0.0793 .
## ---
## Signif. codes:
##   0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1  ' ' 1
##
## Residual standard error: 4.006 on 26 degrees of freedom
## Multiple R-squared:  0.6763, Adjusted R-squared:  0.6389
## F-statistic:  18.1 on 3 and 26 DF,  p-value: 1.501e-06
```

The pre-treatment term is definitely significant, with a P-value of 0.0000025. So pre-treatment score definitely has an impact on post-treatment score.

I didn't ask you to test for significance of drugs. I just wanted you to assess their coefficients. Drug A is being used as the baseline, so its coefficient is zero. Drug D has a slightly positive coefficient (0.109) so its average bacilli score is slightly higher (for any pre-treatment score) than for drug A. Drug F, which was the placebo, has a slope of 3.446, so its average bacilli score is a fair bit higher than for either of the other drugs. This makes sense because a higher score is worse, and the two "real" drugs are both better than the fake one.

Whether there is a real drug difference, I didn't ask you to assess, but you could do it by `drop1` again, this way:

```r
drop1(lepro.2, test = "F")
```

```
## Single term deletions
##
## Model:
## post ~ pre + drug
##        Df Sum of Sq    RSS     AIC F value
## <none>              417.20  86.971
## pre     1    577.90 995.10 111.049 36.0145
## drug    2     68.55 485.76  87.535  2.1361
##           Pr(>F)
```

```
## <none>
## pre    2.454e-06 ***
## drug      0.1384
## ---
## Signif. codes:
##   0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1  ' ' 1
```

This is actually not significant. This is one of those cases where the non-significant `drug` has a slightly *bigger* AIC than `<none>`, so `drop1` considers it best to leave it in the model.

This is why I didn't ask you to test this, since it would have confused the story.

(i) Obtain predicted values for `post` for each of the three drugs at `pre` scores 5, 12 and 20. To do this, obtain a new data frame that has all 9 combinations of drugs and `pre` scores, and then feed this into `predict` using your preferred model.

Analysis of covariance is just a linear model, so *predict* works the same here as in regression.

Solution

First, make the new data frame for predicting from, using `crossing`. I'm doing this in small steps for clarity: first, I define all the drugs and `pre` values, and then I feed them into `crossing`:

```
drugs = c("A", "D", "F")
pres = c(5, 12, 20)
lepro.new = crossing(drug = drugs, pre = pres)
lepro.new
```

```
## # A tibble: 9 x 2
##   drug    pre
##   <chr> <dbl>
## 1 A         5
## 2 A        12
## 3 A        20
## 4 D         5
## 5 D        12
## 6 D        20
## 7 F         5
## 8 F        12
## 9 F        20
```

The names of the columns in the new data frame have to be *exactly* `drug` and `pre`. Anything else, and `predict` won't work.

Now I obtain the predictions, from my best model `lepro.2`. I don't need intervals or anything like that:

```
preds = predict(lepro.2, lepro.new)
preds
```

```
##         1         2         3         4
##  1.055110  7.965396 15.862867  1.164081
##         5         6         7         8
##  8.074368 15.971838  4.501248 11.411535
##         9
## 19.309005
```

only it would be easier to see the predictions side by side with the values they are predictions for:

```
allpreds = cbind(lepro.new, preds)
allpreds
```

```
##   drug pre     preds
## 1    A   5  1.055110
## 2    A  12  7.965396
## 3    A  20 15.862867
## 4    D   5  1.164081
## 5    D  12  8.074368
## 6    D  20 15.971838
## 7    F   5  4.501248
## 8    F  12 11.411535
## 9    F  20 19.309005
```

I gave this a name in case I feel like using it again later.

(j) Now, plot the data with the fitted lines on. You'll need to do the trick of using the original data frame (and plot the points), and the data frame of predictions (and join those points with lines).

Solution

```
ggplot(lepro, aes(x = pre, y = post, colour = drug)) +
    geom_point() + geom_line(data = allpreds,
    aes(y = preds, linetype = drug))
```

Inside the `aes` for the `geom_line`, I only have to specify the things that have changed, so that the `x` (`pre`) and the `colour` (`drug`) are "inherited" unchanged from the first `aes`.

`geom_line` has the `aes` as its default first argument, so I have to specify `data=` by name.

The `linetype=` is really overkill, but I just wanted to show you that you can distinguish the drugs by line type as well.

The line types show up in the legend too, though they're not so easy to see.

(k) Are the lines on your plot parallel, with the same slopes? Is this what you would expect? Explain briefly.

Solution

My lines are parallel. This is exactly what I would expect, since my best model has no interaction, and the interaction is what would make the lines *not* be parallel. If your best model *did* have the interaction term still in it, your predictions would have been these:

```
preds = predict(lepro.1, lepro.new)
cbind(lepro.new, preds)
```

```
##   drug pre      preds
## 1    A   5  2.0955414
## 2    A  12  7.3121019
## 3    A  20 13.2738854
## 4    D   5  0.7572581
## 5    D  12  8.2370968
## 6    D  20 16.7854839
## 7    F   5  2.8637331
## 8    F  12 11.2249823
## 9    F  20 20.7806955
```

The slopes for each drug are no longer the same (because nothing is forcing them to be the same). For example, if you compare drug A

at pre score 5 and 12, it goes up by 5.2, while drug D at the same pre scores goes up by 7.5.

The `ggplot` in this case is actually a lot easier than the other `ggplot`, because the lines are now just the regression lines for each group. So `geom_smooth` will get them:

```
ggplot(lepro, aes(x = pre, y = post, colour = drug)) +
    geom_point() + geom_smooth(method = "lm")
```

The grey intervals are a bit confusing. They are confidence intervals for the mean `post` score (as we did for regression early on in the course). But I left them there to show that they overlap substantially and thus that those slopes are not "really different". That's the lack of interaction. There is, as you see, a substantial scatter in the points that would make it very difficult to prove that those three slopes are really different.

You can get rid of them by saying *se=F* inside the geom-smooth, as normal.

A study was made of the characteristics of urine of young men. The men were classified into four groups based on their degree of obesity. (The groups are labelled `a, b, c, d`.) Four variables were measured, `x` (which you can ignore), pigment creatinine, chloride and chlorine. The data are in link as a `.csv` file. There are 45 men altogether.

(a) Read in the data and check that you have the right number of observations and the right variables.

Solution

`read_csv`:

```
my_url = "http://www.utsc.utoronto.ca/~butler/d29/urine.csv"
urine = read_csv(my_url)
```

```
## Parsed with column specification:
## cols(
##   obesity = col_character(),
##   x = col_integer(),
##   creatinine = col_double(),
##   chloride = col_double(),
##   chlorine = col_double()
## )
```

```
urine
```

```
## # A tibble: 45 x 5
##    obesity     x creatinine chloride chlorine
##    <chr>   <int>      <dbl>    <dbl>    <dbl>
##  1 a          24       17.6     5.15      7.5
##  2 a          32       13.4     5.75      7.1
##  3 a          17       20.3     4.35      2.3
##  4 a          30       22.3     7.55      4
##  5 a          30       20.5     8.5       2
##  6 a          27       18.5    10.2       2
##  7 a          25       12.1     5.95     16.8
##  8 a          30       12       6.3      14.5
##  9 a          28       10.1     5.45      0.9
## 10 a          24       14.7     3.75      2
## # ... with 35 more rows
```

The variables are as promised, and we do indeed have 45 rows.

(b) Make boxplots of each of the three variables of interest against obesity group.

Solution

Just churn through it:

```
ggplot(urine, aes(x = obesity, y = creatinine)) +
    geom_boxplot()
```

```
ggplot(urine, aes(x = obesity, y = chlorine)) +
    geom_boxplot()
```

```
ggplot(urine, aes(x = obesity, y = chloride)) +
    geom_boxplot()
```

This also works with facets, though it is different from the other ones we've done, in that we have to collect together the y-variables first rather than the xs:

```
urine %>% gather(yname, y, creatinine:chlorine) %>%
    ggplot(aes(x = obesity, y = y)) + geom_boxplot() +
    facet_wrap(~yname, scales = "free", ncol = 2)
```

I decided to throw a couple of things in here: first, the `scales="free"` thing since the y-variables (this time) are on different scales, and second, the `ncol=2` to arrange the facets in (3 cells of) a 2×2 grid, rather than having them come out tall and skinny. It's unusual to have faceted boxplots, but this is one of those cases where it makes sense. (The key is different y's but the same x, I think.)

Like one of those crazy drinks at Starbucks.

The conclusions about the boxplots are, of course, the same. I think it makes it easier to have the three boxplots side by side, but it's up to you whether you think that gain is worth the extra coding.

(c) How, if at all, do the groups differ on the variables of interest?

Solution

Any sensible comment here is good. You can take the approach that group D is lowest on creatinine and chloride, but there is not much to choose (given the amount of variability) on chlorine. Or you can say that groups B and C are generally highest, except for chloride where A is higher. Anything of this general kind is fine. The point here is that some of the groups appear to be different on some of the variables, which would make a multivariate analysis of variance (in a moment) come out significant.

(d) Run a multivariate analysis of variance, using the three variables of interest as response variables, and the obesity group as explanatory. (This is a so-called one-way MANOVA.)

Solution

Create the response variable and run `manova`:

```
response = with(urine, cbind(creatinine, chlorine,
    chloride))
urine.1 = manova(response ~ obesity, data = urine)
summary(urine.1)
```

```
##           Df  Pillai approx F num Df den Df
## obesity    3 0.43144   2.2956      9    123
## Residuals 41
##            Pr(>F)
## obesity   0.02034 *
## Residuals
## ---
## Signif. codes:
##   0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1  ' ' 1
```

(e) What do you conclude? Is this in line with what your boxplots said? Explain briefly.

Solution

The null hypothesis (that each of the variables have the same mean for each of the groups) is rejected: that is, not all of the groups have the same mean on all of the variables. Based on the idea that there seemed to be differences between the groups in the boxplots, this makes sense. (That's what I saw, anyway. There was a lot of variability within the groups, which is why the P-value didn't come out smaller.) The other way of doing this is the following, using `Manova` from `car`:

```
response.1 = lm(response ~ obesity, data = urine)
Manova(response.1)
```

```
##
## Type II MANOVA Tests: Pillai test statistic
##         Df test stat approx F num Df den Df
## obesity  3   0.43144   2.2956      9    123
##          Pr(>F)
## obesity 0.02034 *
## ---
## Signif. codes:
##   0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1  ' ' 1
```

The result is the same. You don't need do this one here (though you can as an alternative), but when you come to repeated measures you *will* need to be able to use `Manova`.

To understand the differences in the variables due to the groups, we need to run a discriminant analysis (coming up later).

questionable questions:

A research lab is developing a new compound to relieve severe cases of hay fever. There were two active ingredients, labelled A and B, with each one having three levels, low, medium and high. There were thus $3 \times 3 = 9$ treatment combinations. 36 subjects were available, and these were randomly assigned to combinations of levels of factor A and factor B, so that each combination was tested on 4 people. The response variable was the number of hours of pain relief (so a higher value is better). The data are in link.

(a)[2] Read the data and display its structure. Verify that you have what you were expecting.

Solution

```
my_url = "http://individual.utoronto.ca/kbutler/stad29/hayfever.txt"
hayfever = read.table(my_url, header = T)
hayfever
```

```
##    relief      a      b replicate
## 1     2.4    low    low         1
## 2     2.7    low    low         2
## 3     2.3    low    low         3
## 4     2.5    low    low         4
## 5     4.6    low medium         1
## 6     4.2    low medium         2
## 7     4.9    low medium         3
## 8     4.7    low medium         4
## 9     4.8    low   high         1
## 10    4.5    low   high         2
```

```
## 11    4.4    low   high         3
## 12    4.6    low   high         4
## 13    5.8 medium    low         1
## 14    5.2 medium    low         2
## 15    5.5 medium    low         3
## 16    5.3 medium    low         4
## 17    8.9 medium medium         1
## 18    9.1 medium medium         2
## 19    8.7 medium medium         3
## 20    9.0 medium medium         4
## 21    9.1 medium   high         1
## 22    9.3 medium   high         2
## 23    8.7 medium   high         3
## 24    9.4 medium   high         4
## 25    6.1   high    low         1
## 26    5.7   high    low         2
## 27    5.9   high    low         3
## 28    6.2   high    low         4
## 29    9.9   high medium         1
## 30   10.5   high medium         2
## 31   10.6   high medium         3
## 32   10.1   high medium         4
## 33   13.5   high   high         1
## 34   13.0   high   high         2
## 35   13.3   high   high         3
## 36   13.2   high   high         4
```

(b)[2] interaction plot
Solution

```
hayfever %>% group_by(a, b) %>% summarize(relief.mean = mean(relief)) %>%
    ggplot(aes(x = a, y = relief.mean, colour = b,
        group = b)) + geom_point() + geom_line()
```

(c)[2] Run an analysis of variance with interaction.

Solution

```
hayfever.1 = aov(relief ~ a * b, data = hayfever)
summary(hayfever.1)
```

```
##             Df Sum Sq Mean Sq F value Pr(>F)
## a            2 220.02  110.01  1827.9 <2e-16
## b            2 123.66   61.83  1027.3 <2e-16
## a:b          4  29.42    7.36   122.2 <2e-16
## Residuals   27   1.63    0.06
##
## a           ***
## b           ***
## a:b         ***
## Residuals
## ---
## Signif. codes:
##   0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1  ' ' 1
```

(d)[2] simple effects

One of the first noticeable effects of climate change was "acid rain". This is formed by the water vapour in clouds combining with nitrous oxide and sulfur dioxide, which come from coal and oil production. How does the acidity of rain affect the acidity of the soil on which it falls? (This will have an effect on the kind of plants that can be grown in that soil.) Acidity is measured using the pH scale, where a pH of 7 is chemically neutral, a number less than 7 is acidic, and a number greater than 7 is alkaline.

An experiment was conducted at the Florida Institute of Food and Agricultural Sciences, to determine how acidity of rain affects

soil acidity. Experimental plots were irrigated with rainwater that was prepared to have one of two different pH levels, 3.7 and 4.5. The acidity of the soil was then measured at three different depths, 0–15, 15–30, and 30–46 centimetres. This was done on three different dates, in April and June 1991. The data are in link.

(a)[2] Read in and display the data.

Solution

This time, it's a `.csv`:

```
my_url = "http://www.utsc.utoronto.ca/~butler/d29/acidrain.csv"
acidrain = read_csv(my_url)
```

```
## Parsed with column specification:
## cols(
##   soil_depth = col_character(),
##   date = col_date(format = ""),
##   rain_pH = col_double(),
##   soil_acidity = col_double()
## )
```

```
acidrain
```

```
## # A tibble: 18 x 4
##    soil_depth date       rain_pH soil_acidity
##    <chr>      <date>       <dbl>        <dbl>
##  1 0-15       1991-04-03     3.7         5.33
##  2 0-15       1991-04-03     4.5         5.33
##  3 0-15       1991-06-16     3.7         5.47
##  4 0-15       1991-06-16     4.5         5.47
##  5 0-15       1991-06-30     3.7         5.2
##  6 0-15       1991-06-30     4.5         5.13
##  7 15-30      1991-04-03     3.7         5.27
##  8 15-30      1991-04-03     4.5         5.03
##  9 15-30      1991-06-16     3.7         5.5
## 10 15-30      1991-06-16     4.5         5.53
## 11 15-30      1991-06-30     3.7         5.33
## 12 15-30      1991-06-30     4.5         5.2
## 13 30-46      1991-04-03     3.7         5.37
## 14 30-46      1991-04-03     4.5         5.4
## 15 30-46      1991-06-16     3.7         5.8
## 16 30-46      1991-06-16     4.5         5.6
## 17 30-46      1991-06-30     3.7         5.33
## 18 30-46      1991-06-30     4.5         5.17
```

There are 4 columns, soil depth, date, pH of the rain that was applied (all explanatory) and soil acidity, the response. You'll note

that we have one soil acidity value per combination of the other things (it was a designed experiment).

We're going to ignore the date for now, treating the observations on different dates as replicates.

(b)[3] Draw a grouped boxplot to show how soil acidity depends on the pH of the rain and the soil depth. (The first time you do this you might not get the boxplot you want. How can you fix that?)

Solution

The problem is likely to be that either your `x` or your `fill` for your boxplot is numerical (`rain_pH` is `dbl`) rather than the categorical variable you need. Try to use one of the explanatory variables as `x` and the other one as `fill` (or `colour`):

```
ggplot(acidrain, aes(x = rain_pH, y = soil_acidity,
    fill = soil_depth)) + geom_boxplot()
```

That looks as if it worked, but it didn't. See the boxplot below for how it's supposed to be. I need `x` for the boxplot needs to be categorical. The easiest way to make it such is to wrap it in `factor`:

```
ggplot(acidrain, aes(x = factor(rain_pH), y = soil_acidity,
    fill = soil_depth)) + geom_boxplot()
```

Even though `soil_depth` looks like numbers, the printout of the data frame reveals that it is text, so that is OK.

If you prefer, exchange `x` and `fill`:

```
ggplot(acidrain, aes(fill = factor(rain_pH), y = soil_acidity,
    x = soil_depth)) + geom_boxplot()
```

(c)[2] What does your grouped boxplot say about likely interactions? Explain briefly.

Solution

Think about the effect of one of your explanatory variables, for each level of the other. For example, in the last plot, the effect of the rain pH pm on the soil acidity is very small at the largest and smallest depths, but at the middle soil depth 15–30, the average (median) soil acidity is a lot less when the rain pH is *higher* (which seems odd to me). The effect of rain pH being different according to soil pH is what suggests an interaction effect. Another way of thinking

about this is imagining what an interaction plot would look like. This would be a trace going through the middle (strictly, mean rather than median) of each set of boxplots of one colour. In the last boxplot, the red trace would go close to straight across, while the blue one would dip in the middle. Not parallel, so suggesting an interaction. Either approach is good.

(d)[3] Fit an ANOVA with interaction. What do you conclude from it? You may want to create a column that is the factor version of `rain_pH` first.

Solution

Following my own hint:

```
acidrain = acidrain %>% mutate(frph = factor(rain_pH))
soil.1 = aov(soil_acidity ~ frph * soil_depth,
    data = acidrain)
summary(soil.1)
```

```
##                 Df Sum Sq Mean Sq F value
## frph             1 0.0304 0.03042   0.759
## soil_depth       2 0.0671 0.03357   0.838
## frph:soil_depth  2 0.0078 0.00391   0.097
## Residuals       12 0.4810 0.04008
##                 Pr(>F)
## frph             0.401
## soil_depth       0.457
## frph:soil_depth  0.908
## Residuals
```

The interaction is nowhere near significant, perhaps surprisingly. But bear in mind that there are only 18 observations in total, so each box on the boxplot is based on *three* observations only. So the interaction would have to be a lot bigger to be significant.

The usual procedure after finding a significant interaction is to take it out:

```
soil.2 = update(soil.1, . ~ . - frph:soil_depth)
summary(soil.2)
```

```
##             Df Sum Sq Mean Sq F value Pr(>F)
## frph         1 0.0304 0.03042   0.871  0.366
## soil_depth   2 0.0671 0.03357   0.962  0.406
## Residuals   14 0.4888 0.03492
```

The P-values have come down a bit (the result of gaining two df for error while the error SS only got a tiny bit bigger), but not nearly enough to be significant.