Assignment 6

Due Thursday November 1 at 11:59pm on Blackboard

This is very long. Do not be alarmed. There are two questions to hand in, right at the end.

The assignment is due on the date shown above. An assignment handed in after the deadline is late, and may or may not be accepted (see course outline). My solutions to the assignment questions will be available when everyone has handed in their assignment.

You are reminded that work handed in with your name on it must be *entirely your own work*.

Assignments are to be handed in on Quercus. See `https://www.utsc.utoronto.ca/~butler/c32/quercus1.nb.html` for instructions on handing in assignments in Quercus. Markers' comments and grades will be available there as well.

For this assignment, hand in *one* HTML document containing both your answers to the questions about the choice of location for the Mexican restaurant, and your report about the Boston Marathon. In your report, you might use a `#` heading for the title, and `##` headings for any sections it contains.

Begin with the usual:

```
library(tidyverse)
```

1. Do adult deer eat different amounts of food at different times of the year? The data in `http://www.utsc.utoronto.ca/~butler/c32/deer.txt` 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
```

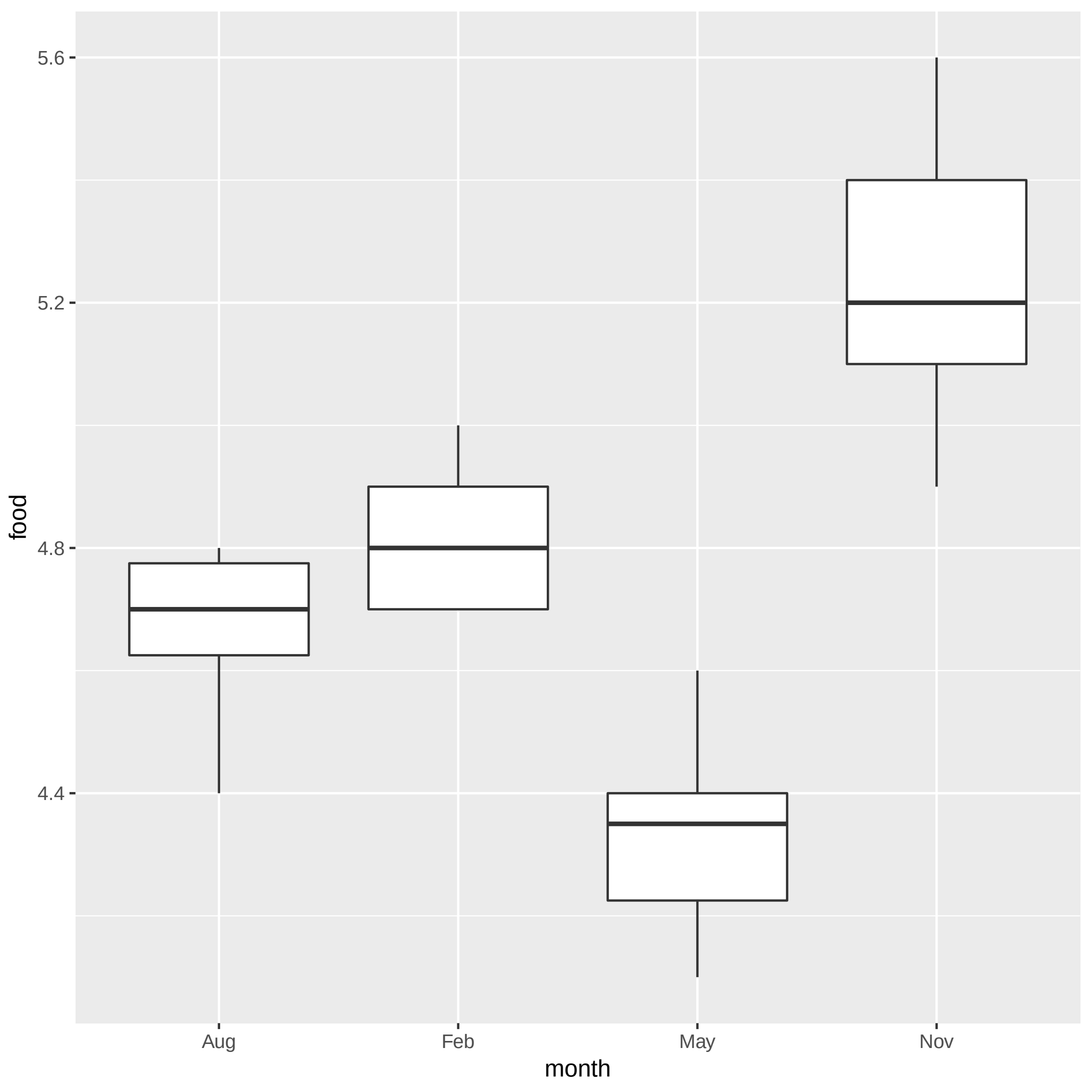
When you want the number of observations *plus* some other summaries, as here, the group-by and summarize idea is the way, using `n()` to get the number of observations in each group. `count` counts the number of observations per group when you *only* have grouping variables.

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()
```

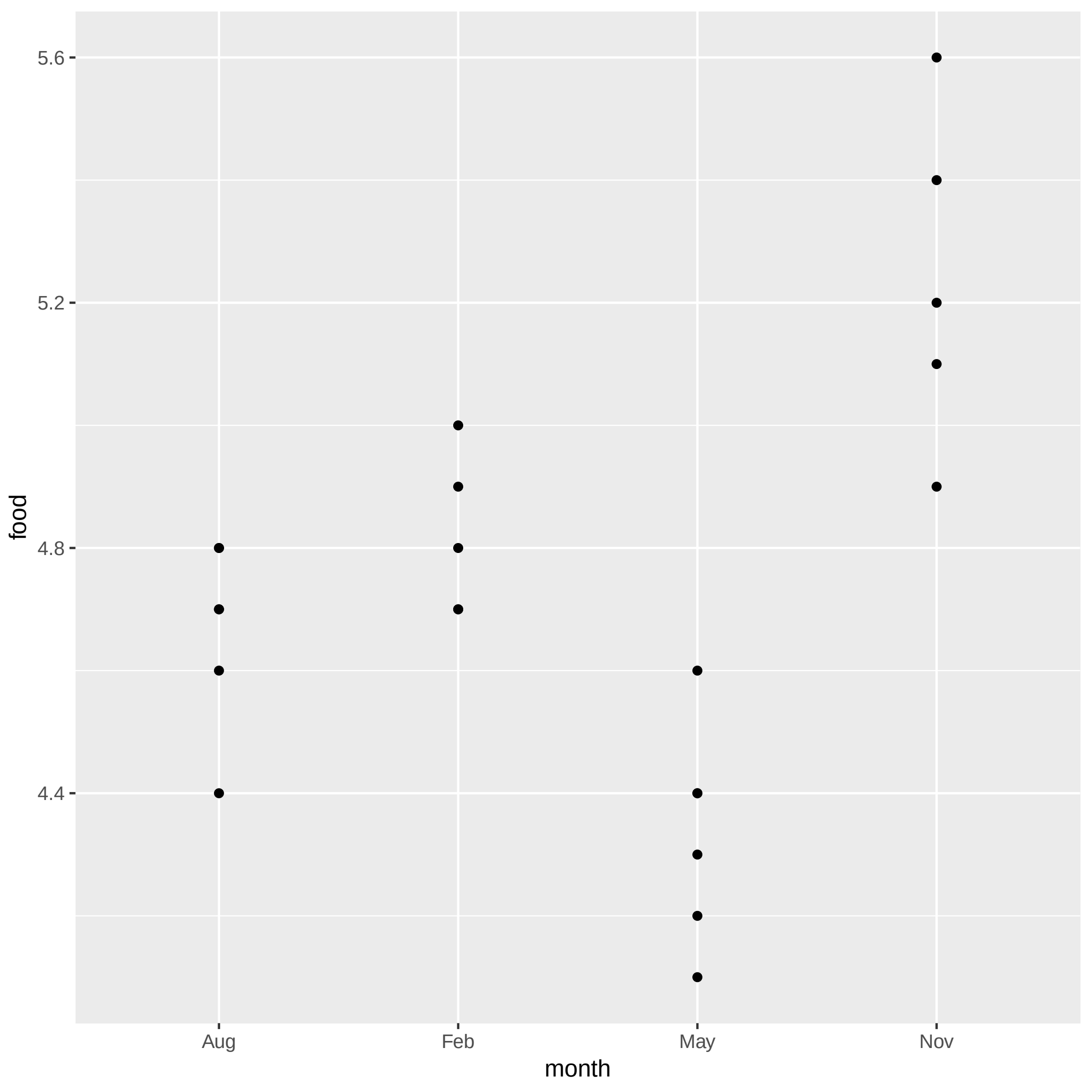


This offers the suggestion that maybe November will be significantly higher than the rest and May significantly lower, or at least they will be significantly different from each other.

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:

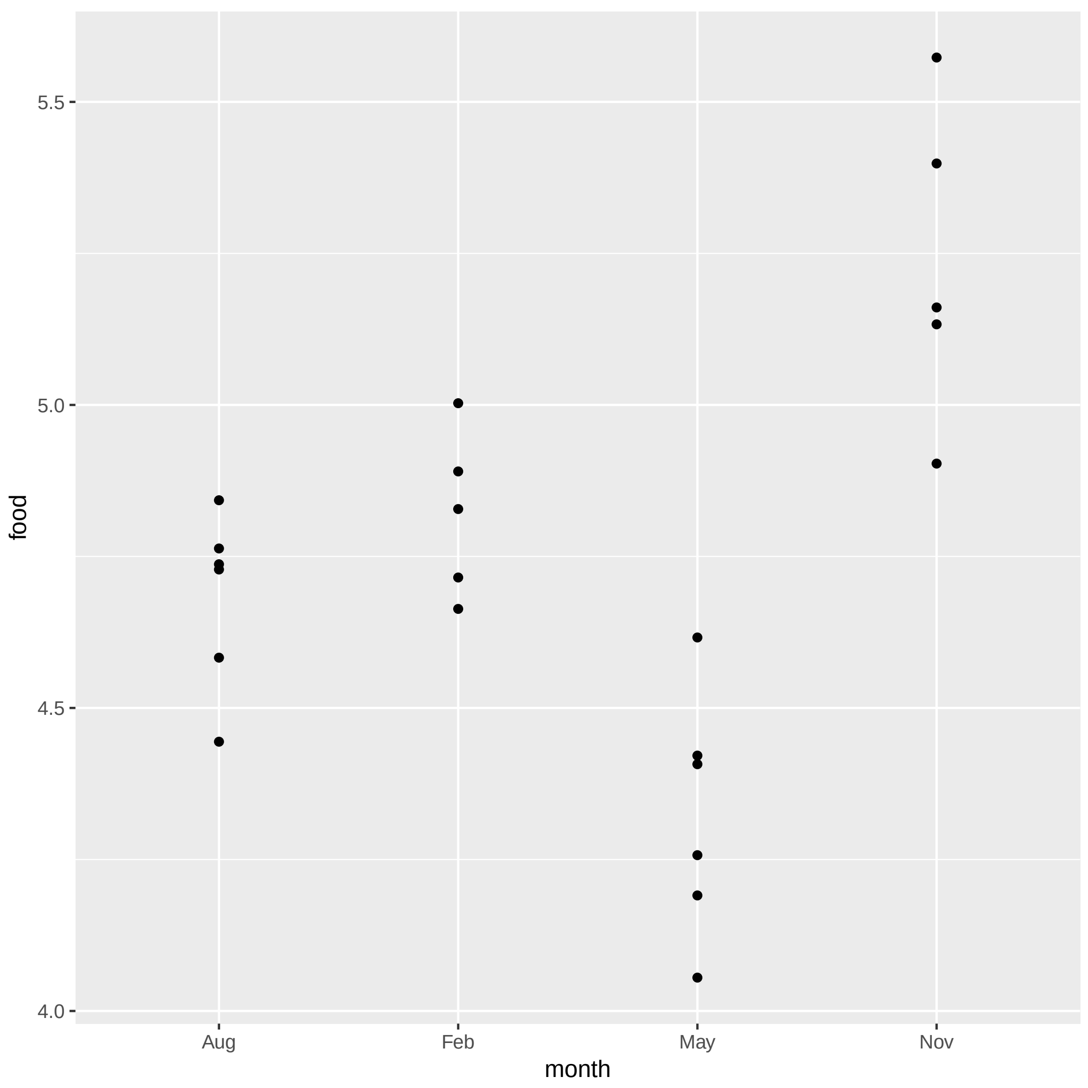
```
ggplot(deer,aes(x=month,y=food))+geom_point()
```



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:

```
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.[1]

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   Pr(>F)
## month        3 2.3065  0.7688   22.08 2.94e-06 ***
## Residuals   18 0.6267  0.0348
## ---
## 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     p adj
## Feb-Aug  0.1533333 -0.16599282  0.4726595 0.5405724
## May-Aug -0.3333333 -0.63779887 -0.0288678 0.0290758
## Nov-Aug  0.5733333  0.25400718  0.8926595 0.0004209
## May-Feb -0.4866667 -0.80599282 -0.1673405 0.0021859
## Nov-Feb  0.4200000  0.08647471  0.7535253 0.0109631
## Nov-May  0.9066667  0.58734052  1.2259928 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.[2]

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.

2. This question again uses the movie rating data at `http://www.utsc.utoronto.ca/~butler/c32/movie-lengths.csv`.

   (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))
```

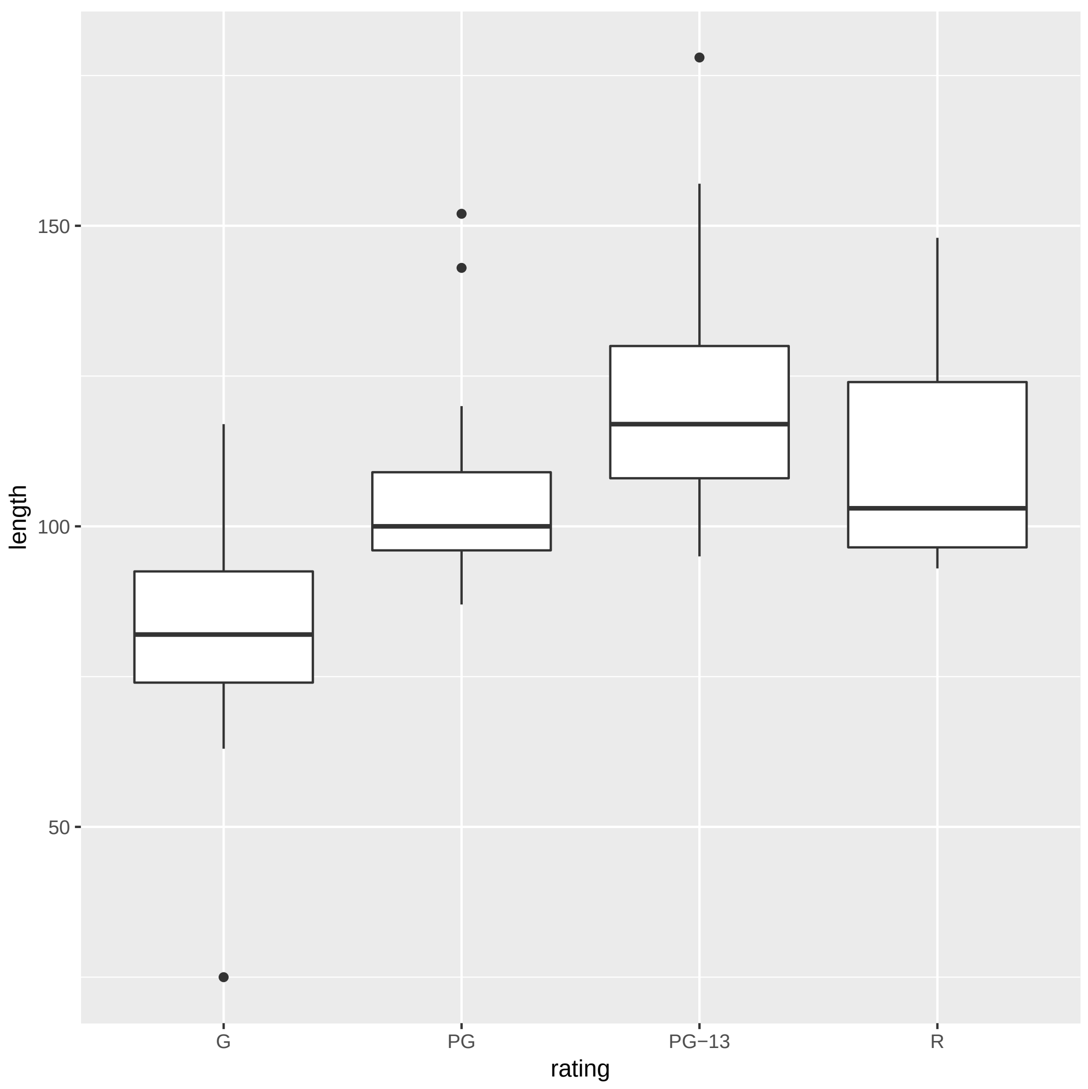
```
## # A tibble: 4 x 3
##   rating count   med
##   <chr>  <int> <int>
## 1 G         15    82
## 2 PG        15   100
## 3 PG-13     15   117
## 4 R         15   103
```

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()
```



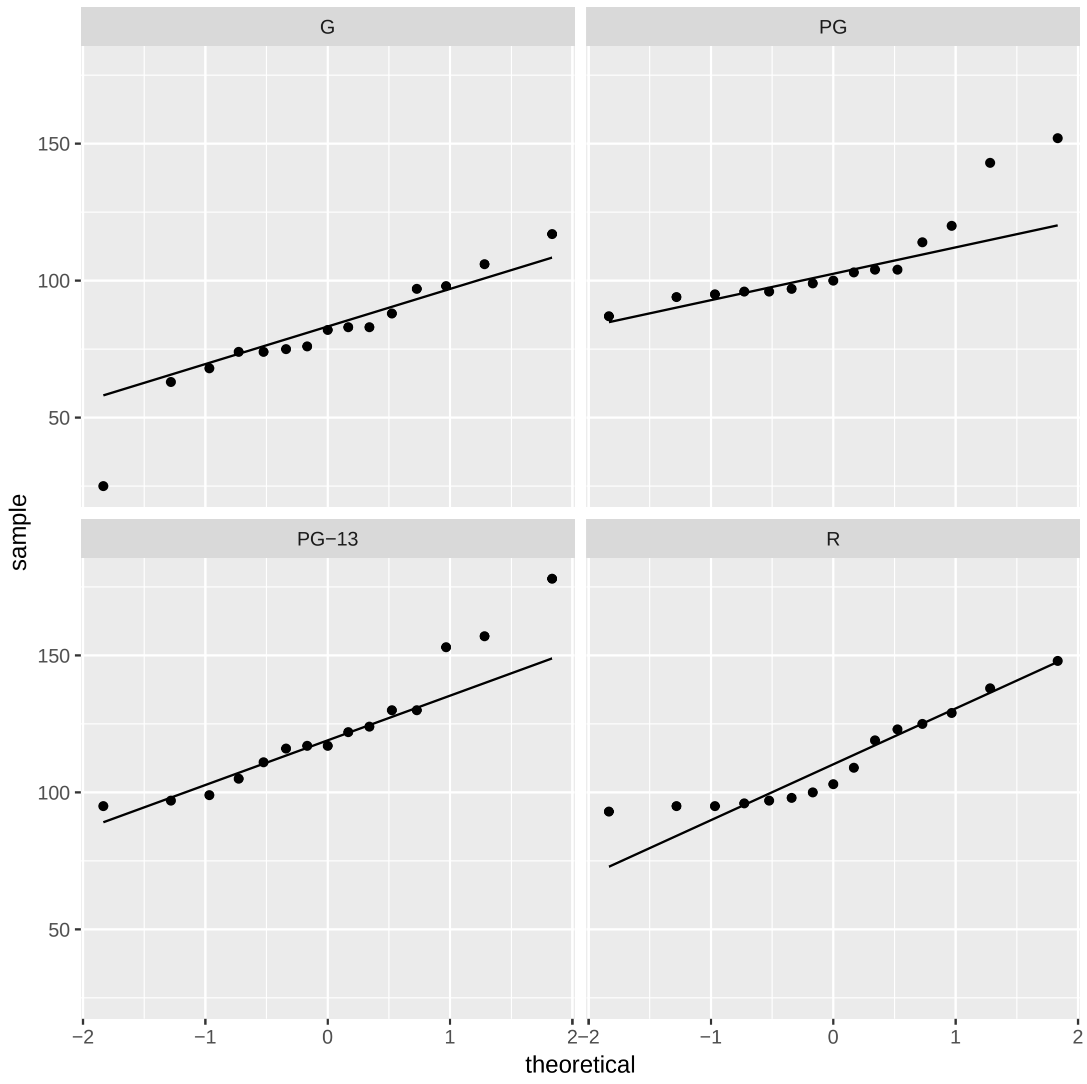
We are looking for approximate normal distributions with approximately equal spreads, which I don't think we have: there are outliers, at the low end for G movies, and at the high end for PG and PG-13 movies. Also, you might observe that the distribution of lengths for R movies is skewed to the right. (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 $\leq$ (`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`.[3]

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[4]) is exactly what the `map2` family of functions does. In our case, `comp2` returns a decimal number, a `dbl`, so `map2_dbl` does it. Thus:

```
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:[5]

```
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: `https://raw.githubusercontent.com/nxskok/smmr/master/R/pairwise_median_test.R`. 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.

Atomic weight of carbon

3. 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 `http://www.utsc.utoronto.ca/~butler/c32/carbon.txt`. 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:

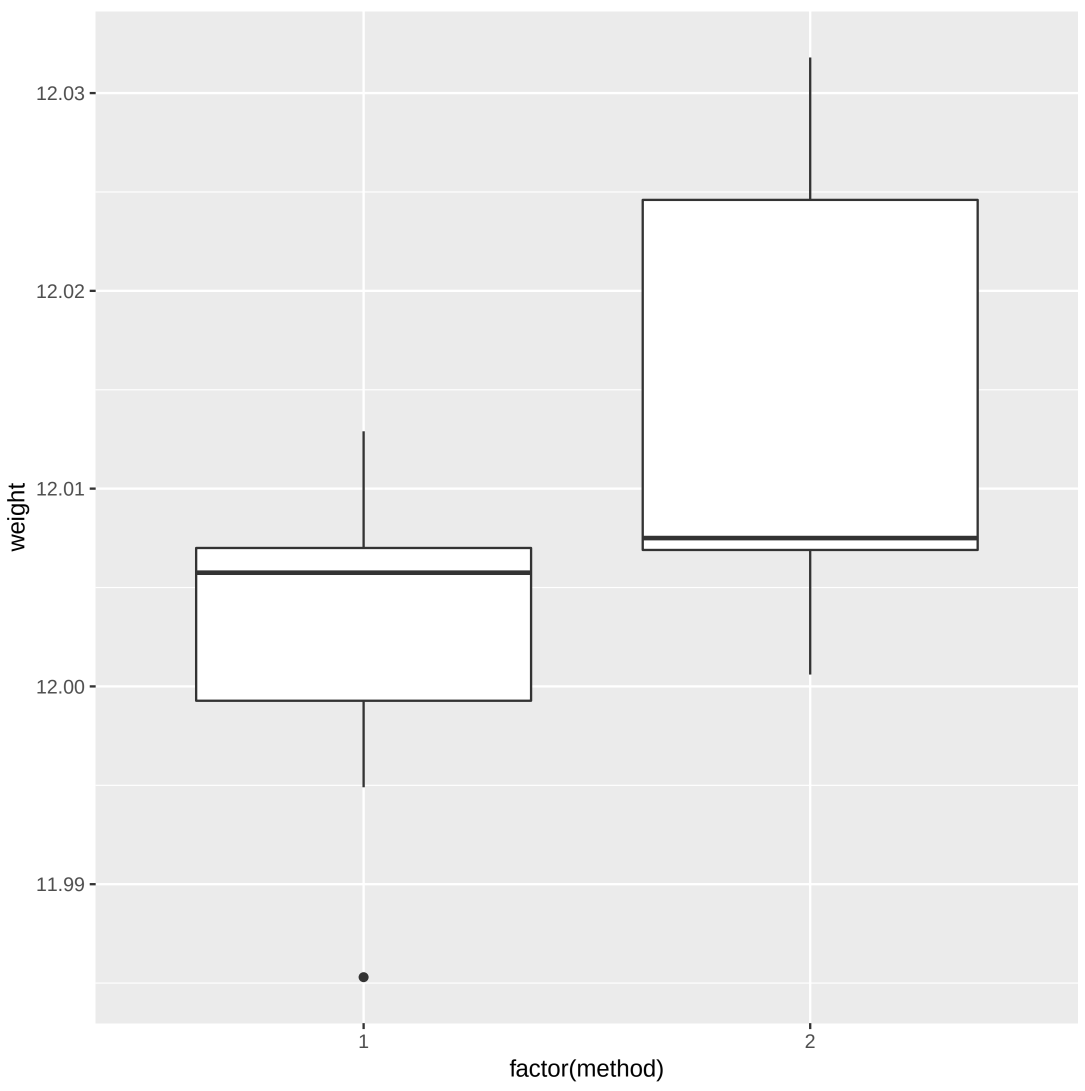
```
my_url="http://www.utsc.utoronto.ca/~butler/c32/carbon.txt"
carbon=read_delim(my_url," ")

## 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[6] of atomic weights close to the nominal figure of 12.

(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.)

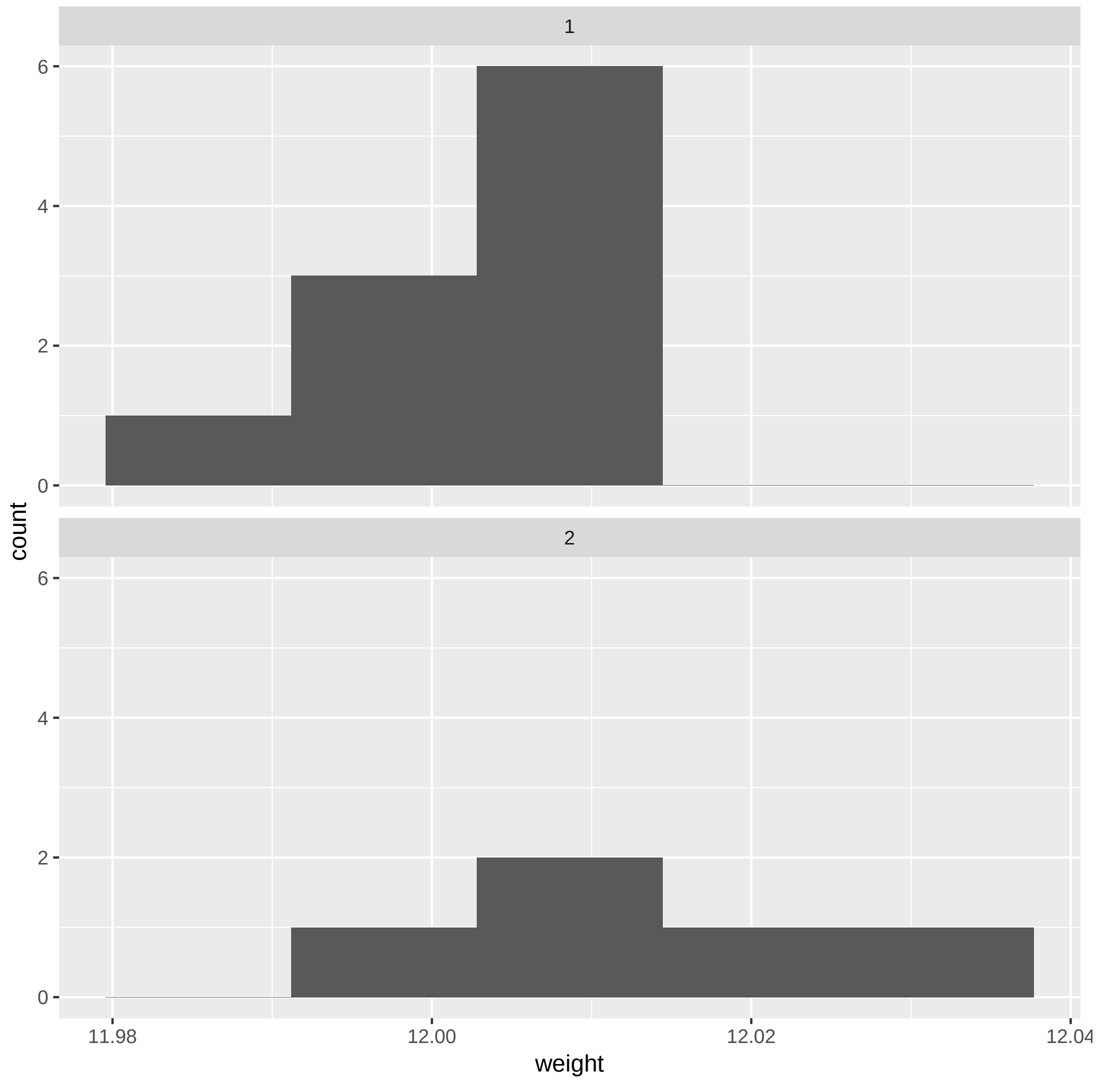
Solution: The appropriate plot, with a categorical method and quantitative weight, is something like a boxplot. If you're not careful, `method` will get treated as a quantitative variable, which you don't want; the easiest way around that, for a boxplot at least, is to turn it into a factor like this:

```
ggplot(carbon,aes(x=factor(method),y=weight))+geom_boxplot()
```



If you insist, you could do a faceted histogram (above and below, for preference):

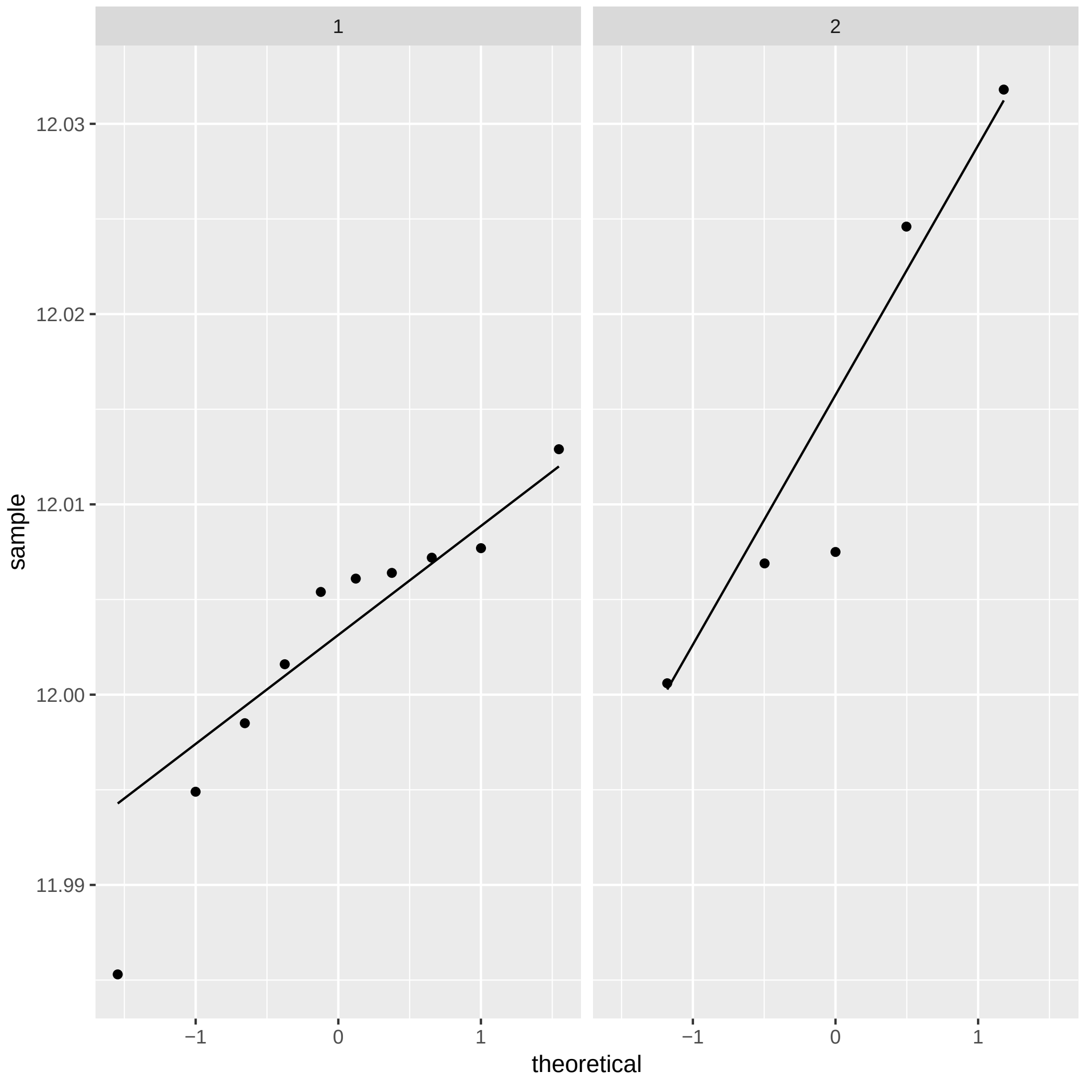
```
ggplot(carbon,aes(x=weight))+geom_histogram(bins=5)+
  facet_wrap(~method,ncol=1)
```



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:

```r
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:

```
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

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.[7]

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.[8]

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".[9] 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 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:

```r
detach(package:car,unload=T)
```

Yet more questions:

4. This question is about the Blue Jays data set (that I used in class).

   (a) The Blue Jays baseball data set is at `http://www.utsc.utoronto.ca/~butler/c32/jays15-home.csv`. 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:

```
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   result  runs Oppruns innings
##    <int> <int> <chr> <chr> <chr> <chr> <chr> <chr>  <int>   <int>   <int>
##  1    82     7 Mond~ boxs~ TOR   <NA>  TBR   L          1       2      NA
##  2    83     8 Tues~ boxs~ TOR   <NA>  TBR   L          2       3      NA
##  3    84     9 Wedn~ boxs~ TOR   <NA>  TBR   W         12       7      NA
##  4    85    10 Thur~ boxs~ TOR   <NA>  TBR   L          2       4      NA
##  5    86    11 Frid~ boxs~ TOR   <NA>  ATL   L          7       8      NA
##  6    87    12 Satu~ boxs~ TOR   <NA>  ATL   W-wo       6       5      10
##  7    88    13 Sund~ boxs~ TOR   <NA>  ATL   L          2       5      NA
##  8    89    14 Tues~ boxs~ TOR   <NA>  BAL   W         13       6      NA
##  9    90    15 Wedn~ boxs~ TOR   <NA>  BAL   W          4       2      NA
## 10    91    16 Thur~ boxs~ TOR   <NA>  BAL   W          7       6      NA
## # ... with 15 more rows, and 10 more variables: 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.[10]

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

```
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:

```
jays %>% filter(opp=="NYY") %>% print(width=Inf)

## # A tibble: 3 x 21
##     row  game date                box      team  venue opp   result  runs
##   <int> <int> <chr>               <chr>    <chr> <chr> <chr> <chr>  <int>
## 1    92    27 Monday, May 4    boxscore TOR   <NA>  NYY   W          3
## 2    93    28 Tuesday, May 5   boxscore TOR   <NA>  NYY   L          3
## 3    94    29 Wednesday, May 6 boxscore TOR   <NA>  NYY   W          5
##   Oppruns innings wl    position gb    winner  loser    save   `game time`
##     <int>   <int> <chr>    <int> <chr> <chr>   <chr>    <chr>  <time>
## 1       1      NA 13-14        4 3.5   Dickey  Martin   Cecil  02:18
## 2       6      NA 13-15        5 4.5   Pineda  Estrada  Miller 02:54
## 3       1      NA 14-15        3 3.5   Buehrle Sabathia <NA>   02:30
##   Daynight attendance streak
##   <chr>         <int> <chr>
## 1 N             19217 +
## 2 N             21519 -
## 3 N             21312 +
```

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.[11] 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.

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  venue opp   result  runs
##   <int> <int> <chr>              <chr>    <chr> <chr> <chr> <chr>  <int>
## 1    92    27 Monday, May 4      boxscore TOR   <NA>  NYY   W          3
## 2    93    28 Tuesday, May 5     boxscore TOR   <NA>  NYY   L          3
## 3    94    29 Wednesday, May 6 boxscore TOR   <NA>  NYY   W          5
##   Oppruns innings
##     <int>   <int>
## 1       1      NA
## 2       6      NA
## 3       1      NA

jays %>% filter(opp=="NYY") %>% select(wl:streak) %>% print(width=Inf)

## # A tibble: 3 x 10
##   wl    position gb    winner  loser    save   `game time` Daynight
##   <chr>    <int> <chr> <chr>   <chr>    <chr>  <time>      <chr>
## 1 13-14        4 3.5   Dickey  Martin   Cecil  02:18       N
## 2 13-15        5 4.5   Pineda  Estrada  Miller 02:54       N
## 3 14-15        3 3.5   Buehrle Sabathia <NA>   02:30       N
##   attendance streak
##        <int> <chr>
## 1      19217 +
## 2      21519 -
## 3      21312 +
```

(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
```

The column names do not need quotes; this is part of the nice stuff about `dplyr`. 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.
```

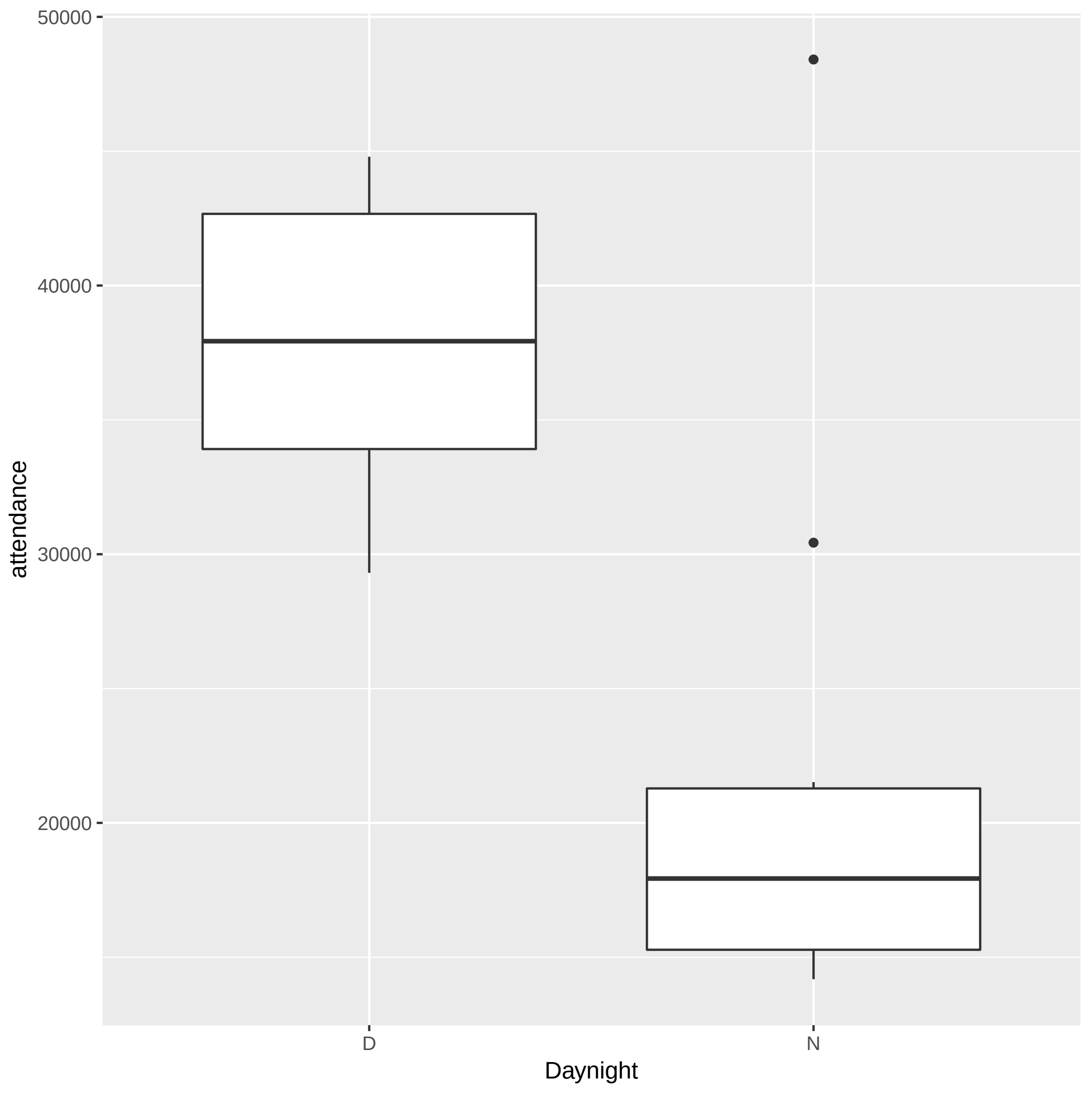
The mean attendances are about 38 thousand and about 20 thousand. Note that the night games have much the larger SD, possibly because of the large outlier night attendance (opening night). Which we can also investigate.

```
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
```

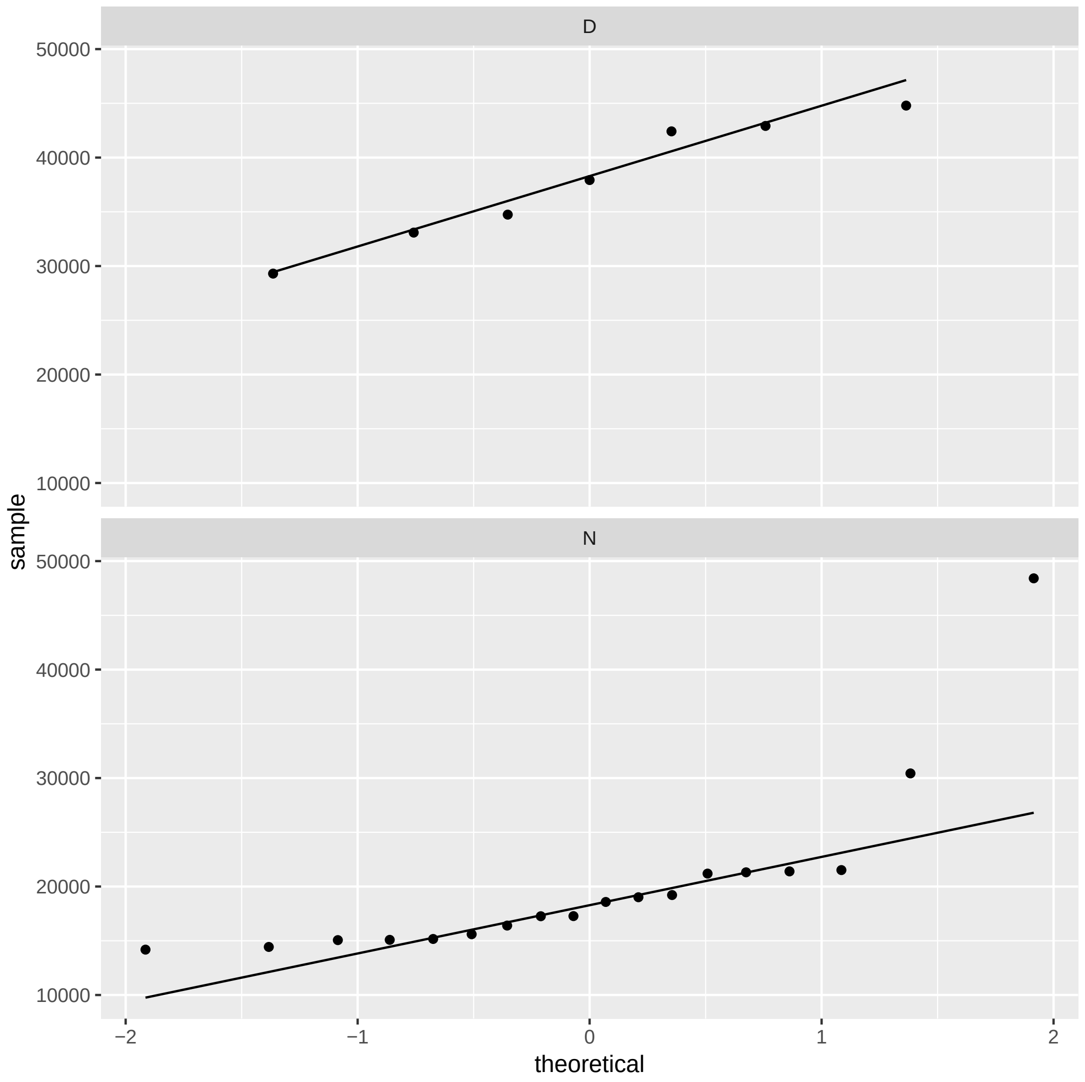
There was one attendance exactly equal to the overall median (as you would expect: with an odd number of data values, the median is one of the data values). `smmr` removed it; if you did the test by hand, what happens to it depends on whether you counted aboves or belows, and this will have a small effect 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.[12]

(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.

5. 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 `http://www.utsc.utoronto.ca/~butler/c32/ratsleep.txt`.

   (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 `http://www.utsc.utoronto.ca/~butler/c32/ratsleep2.txt`, 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:

```
(sleep1 %>% gather(rep,sleeptime,obs1:obs5) -> sleep)
```

```
## # 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
```

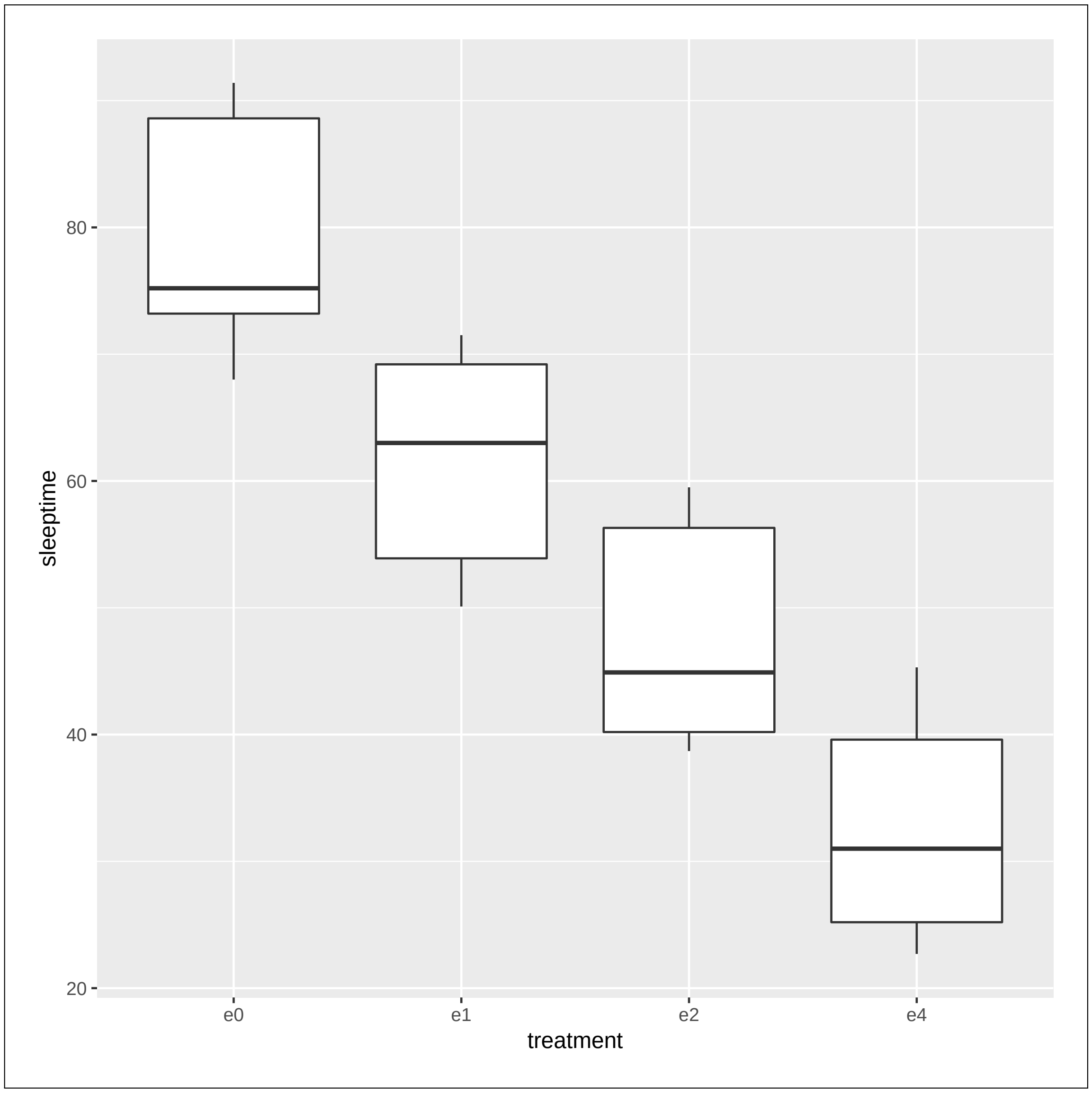
We have 20 rows of 3 columns. I got all the rows, but you will probably get an output with ten rows as usual, and will need to click Next to see the last ten rows. The initial display will say how many rows (20) and columns (3) you have.

The column `rep` is not very interesting: it just says which observation each one was within its group.[13] The interesting things are `treatment` and `sleeptime`, which are the two variables we'll need for our analysis of variance.

(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   Pr(>F)
## treatment    3   5882    1961   21.09 8.32e-06 ***
## Residuals   16   1487      93
## ---
## 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.

6. 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 `http://www.utsc.utoronto.ca/~butler/c32/tomatoes.txt`.

(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
```

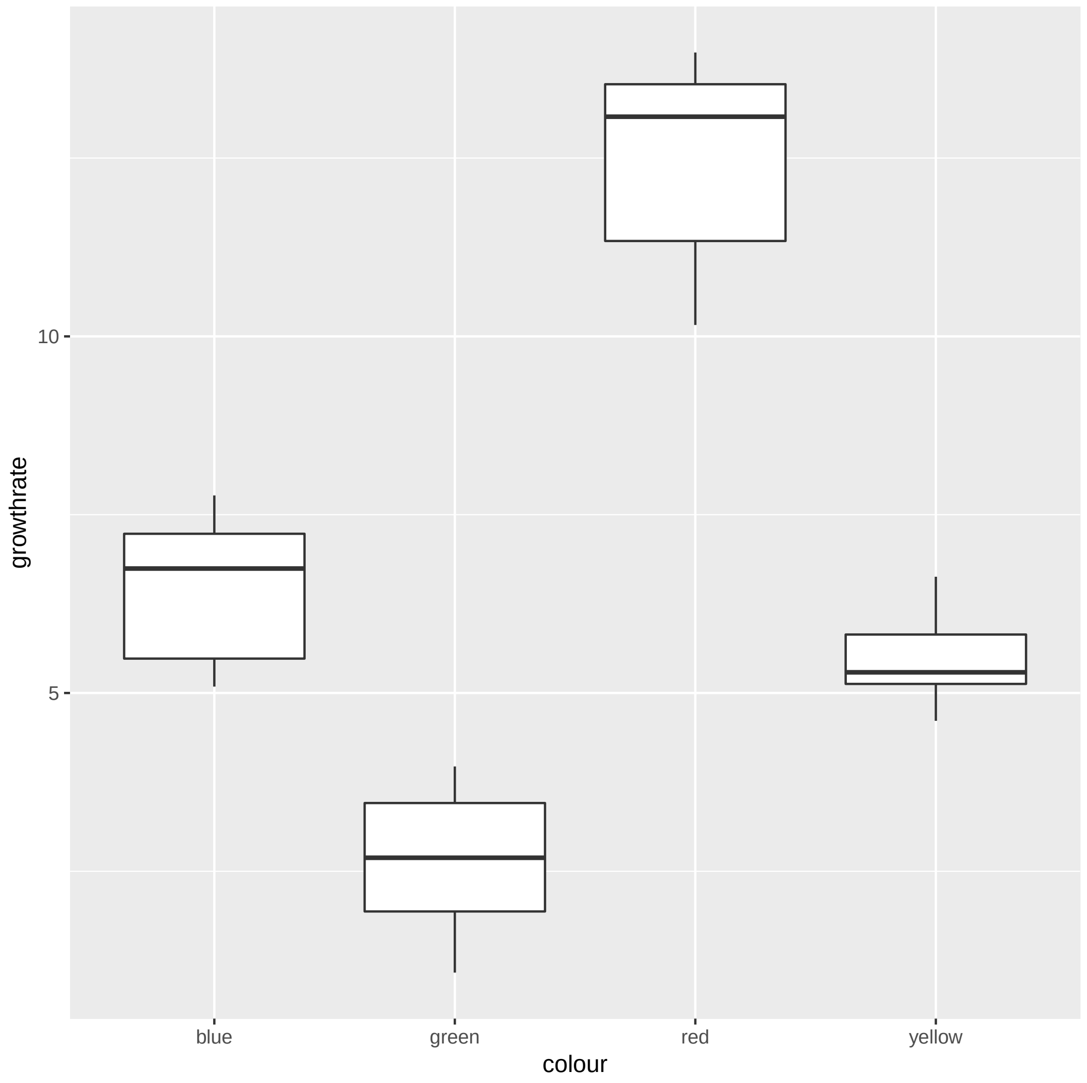
```
## plant,colour,growthrate
## 1,blue,5.34
## 2,blue,7.45
## 3,blue,7.15
## 4,blue,5.53
## 5,blue,6.34
## 6,blue,7.16
## 7,blue,7.77
## 8,blue,5.09
## 1,red,13.67
## 2,red,13.04
## 3,red,10.16
## 4,red,13.12
## 5,red,11.06
## 6,red,11.43
## 7,red,13.98
## 8,red,13.49
## 1,yellow,4.61
## 2,yellow,6.63
## 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)

##
## 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   Pr(>F)
## colour       3  410.5  136.82   118.2 5.28e-16 ***
## Residuals   28   32.4    1.16
## ---
## 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     p adj
## green-blue   -3.8125 -5.281129 -2.3438706 0.0000006
## red-blue      6.0150  4.546371  7.4836294 0.0000000
## yellow-blue  -0.9825 -2.451129  0.4861294 0.2825002
## red-green     9.8275  8.358871 11.2961294 0.0000000
## yellow-green  2.8300  1.361371  4.2986294 0.0000766
## yellow-red   -6.9975 -8.466129 -5.5288706 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.

7. The data in `http://www.utsc.utoronto.ca/~butler/c32/migraine.txt` 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.[14]

(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:

```
(migraine %>% gather(drug,painrelief,DrugA:DrugC) -> migraine2)

## # 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  Pr(>F)
## drug         2  41.19   20.59   7.831 0.00241 **
## Residuals   24  63.11    2.63
## ---
## 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     p adj
## DrugB-DrugA  2.8888889  0.9798731 4.797905 0.0025509
## DrugC-DrugA  2.2222222  0.3132065 4.131238 0.0203671
## DrugC-DrugB -0.6666667 -2.5756824 1.242349 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  Pr(>F)
## drug         2  41.19   20.59   7.831 0.00241 **
## Residuals   24  63.11    2.63
## ---
## 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`.[15] I also used the `data=.` trick to use "the data frame that came out of the previous step" as my input to `aov`.

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     p adj
## DrugB-DrugA  2.8888889  0.9798731 4.797905 0.0025509
## DrugC-DrugA  2.2222222  0.3132065 4.131238 0.0203671
## DrugC-DrugB -0.6666667 -2.5756824 1.242349 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  Pr(>F)
## drug          2  41.19   20.59   7.831 0.00241 **
## Residuals    24  63.11    2.63
## ---
## 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     p adj
## DrugB-DrugA  2.8888889  0.9798731 4.797905 0.0025509
## DrugC-DrugA  2.2222222  0.3132065 4.131238 0.0203671
## DrugC-DrugB -0.6666667 -2.5756824 1.242349 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`[16] 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`.

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))
```

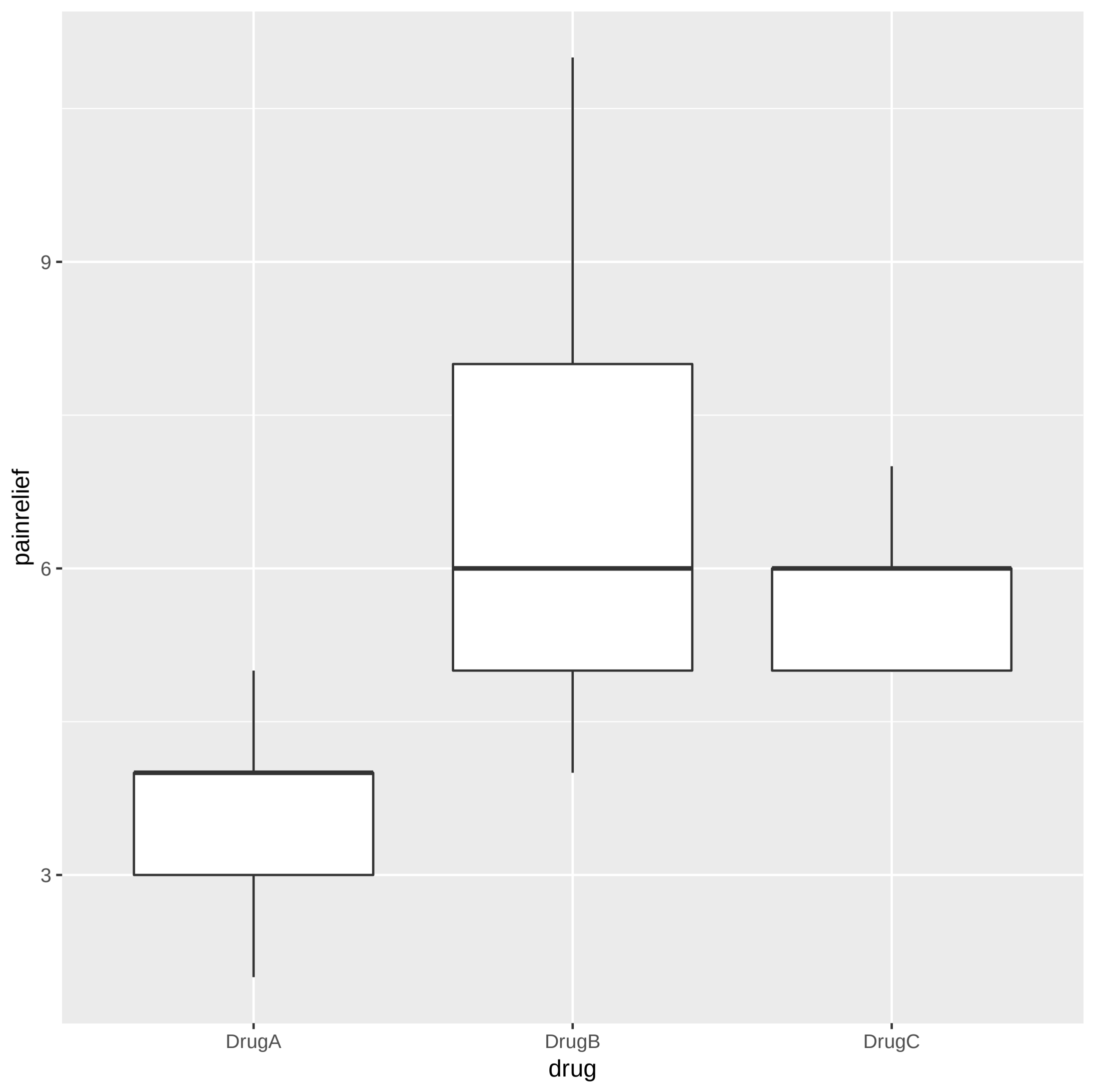
```
## # A tibble: 3 x 2
##   drug      m
##   <chr> <dbl>
## 1 DrugA  3.67
## 2 DrugB  6.56
## 3 DrugC  5.89
```

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.[17]

Another way is to draw a boxplot of pain-relief scores:

```
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!

8. 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 `http://www.utsc.utoronto.ca/~butler/c32/disease.txt`. 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 %>% gather(disloc,frequency,px:ay) -> tbl.2)
```

```
## # 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 %>% gather(disloc,frequency,-Species) -> tbl.2)
```

```
## # 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.2 %>% separate(disloc,c("disease","location"),1) -> tbl.3)
```

```
## # 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):

```
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      LRT Pr(>Chi)
## <none>                      0.000000 55.291
## Species:location:disease  1 0.070257 53.362 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  Pr(>Chi)
## <none>                0.0703 53.362
## Species:location  1  13.0627 64.354 12.9924 0.0003128 ***
## Species:disease   1   0.2696 51.561  0.1993 0.6552865
## location:disease  1   0.1043 51.396  0.0340 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  Pr(>Chi)
## <none>                0.1043 51.396
## Species:location  1  13.0678 62.359 12.9635 0.0003176 ***
## Species:disease   1   0.2746 49.566  0.1703 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:

```
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  Pr(>Chi)
## <none>                0.2746 49.566
## disease           1   2.3617 49.653  2.0871 0.1485461
## Species:location  1  13.2381 60.530 12.9635 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:

```
model.5=update(model.4,.~.-disease)
drop1(model.5,test="Chisq")
```

```
## Single term deletions
##
## Model:
## frequency ~ Species + location + Species:location
##                  Df Deviance    AIC    LRT  Pr(>Chi)
## <none>                2.3617 49.653
## Species:location  1  15.3252 60.617 12.963 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):

```
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  Pr(>Chi)
## <none>                0.000 28.400
## disease:Species  1   15.518 41.918 15.518 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.)

9. My cars data file can be found at `http://www.utsc.utoronto.ca/~butler/c32/cars.csv`. 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
##    <chr>              <dbl>  <dbl>     <int> <int> <chr>
##  1 Buick Skylark       28.4   2.67         4    90 U.S.
##  2 Dodge Omni          30.9   2.23         4    75 U.S.
##  3 Mercury Zephyr      20.8   3.07         6    85 U.S.
##  4 Fiat Strada         37.3   2.13         4    69 Italy
##  5 Peugeot 694 SL      16.2   3.41         6   133 France
##  6 VW Rabbit           31.9   1.92         4    71 Germany
##  7 Plymouth Horizon    34.2   2.2          4    70 U.S.
##  8 Mazda GLC           34.1   1.98         4    65 Japan
##  9 Buick Estate Wagon  16.9   4.36         8   155 U.S.
## 10 Audi 5000           20.3   2.83         5   103 Germany
## # ... 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 Skylark       28.4   2.67         4 U.S.
##  2 Dodge Omni          30.9   2.23         4 U.S.
##  3 Mercury Zephyr      20.8   3.07         6 U.S.
##  4 Fiat Strada         37.3   2.13         4 Italy
##  5 Peugeot 694 SL      16.2   3.41         6 France
##  6 VW Rabbit           31.9   1.92         4 Germany
##  7 Plymouth Horizon    34.2   2.2          4 U.S.
##  8 Mazda GLC           34.1   1.98         4 Japan
##  9 Buick Estate Wagon  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 Buick Estate Wagon         16.9   4.36         8   155 U.S.
## 2 Chevy Malibu Wagon         19.2   3.60         8   125 U.S.
## 3 Chrysler LeBaron Wagon     18.5   3.94         8   150 U.S.
## 4 Ford LTD                   17.6   3.72         8   129 U.S.
## 5 Dodge St Regis             18.2   3.83         8   135 U.S.
## 6 Ford Country Squire Wagon  15.5   4.05         8   142 U.S.
## 7 Mercury Grand Marquis      16.5   3.96         8   138 U.S.
## 8 Chevy Caprice Classic      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.

Salaries of social workers

10. 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 `http://www.utsc.utoronto.ca/~butler/c32/socwork.txt`.

    (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 salary_max
##            <dbl>      <dbl>          <dbl>      <dbl>
## 1              1      16105             28      99139
```

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:

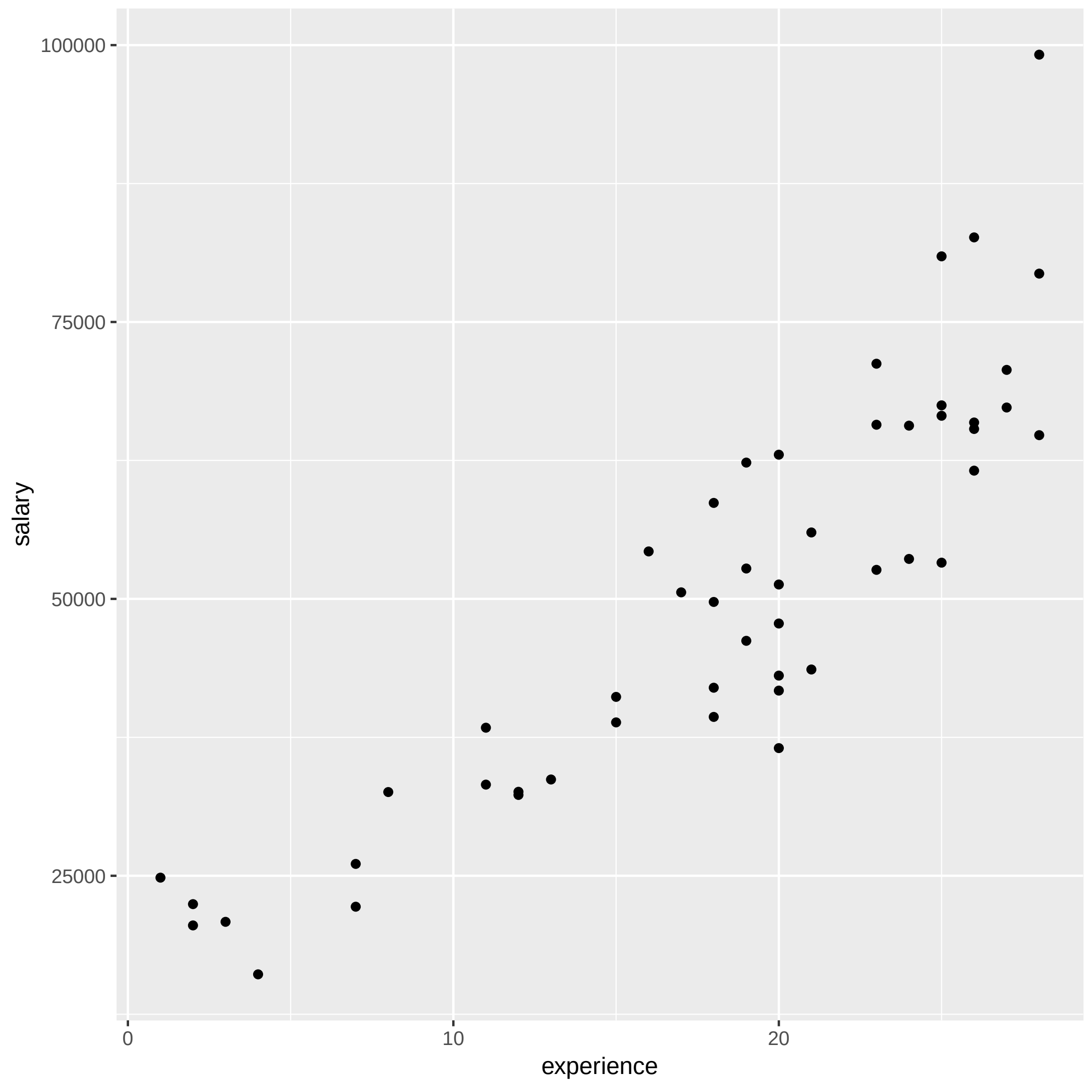
```
soc %>% summarize_all("median")
```

```
## # 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)
```

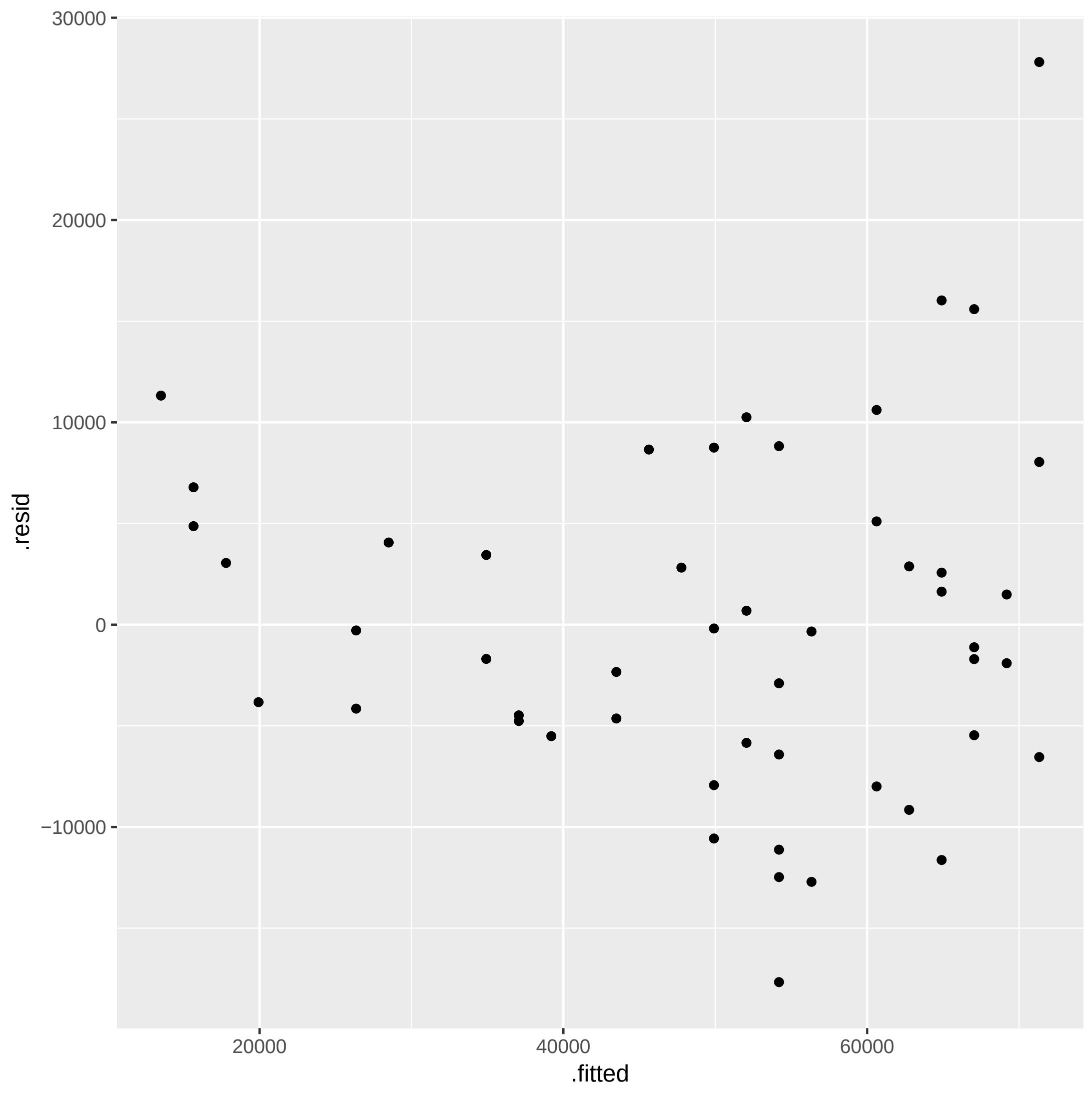
```
##
## Call:
## lm(formula = salary ~ experience, data = soc)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -17666.3  -5498.2   -726.7   4667.7  27811.6
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  11368.7     3160.3   3.597 0.000758 ***
## experience    2141.4      160.8  13.314  < 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:

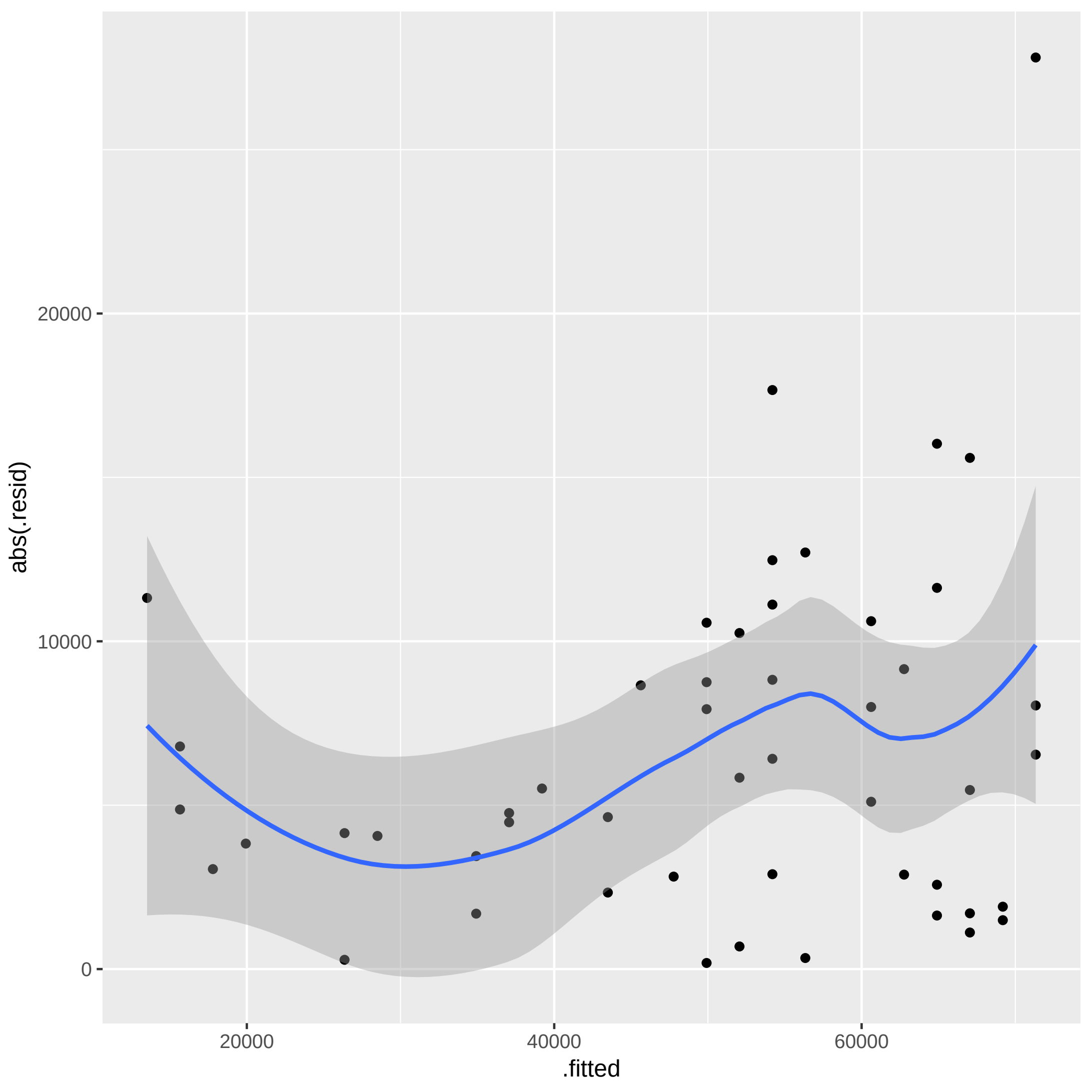
```
ggplot(soc.1,aes(x=.fitted,y=.resid))+geom_point()
```



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.

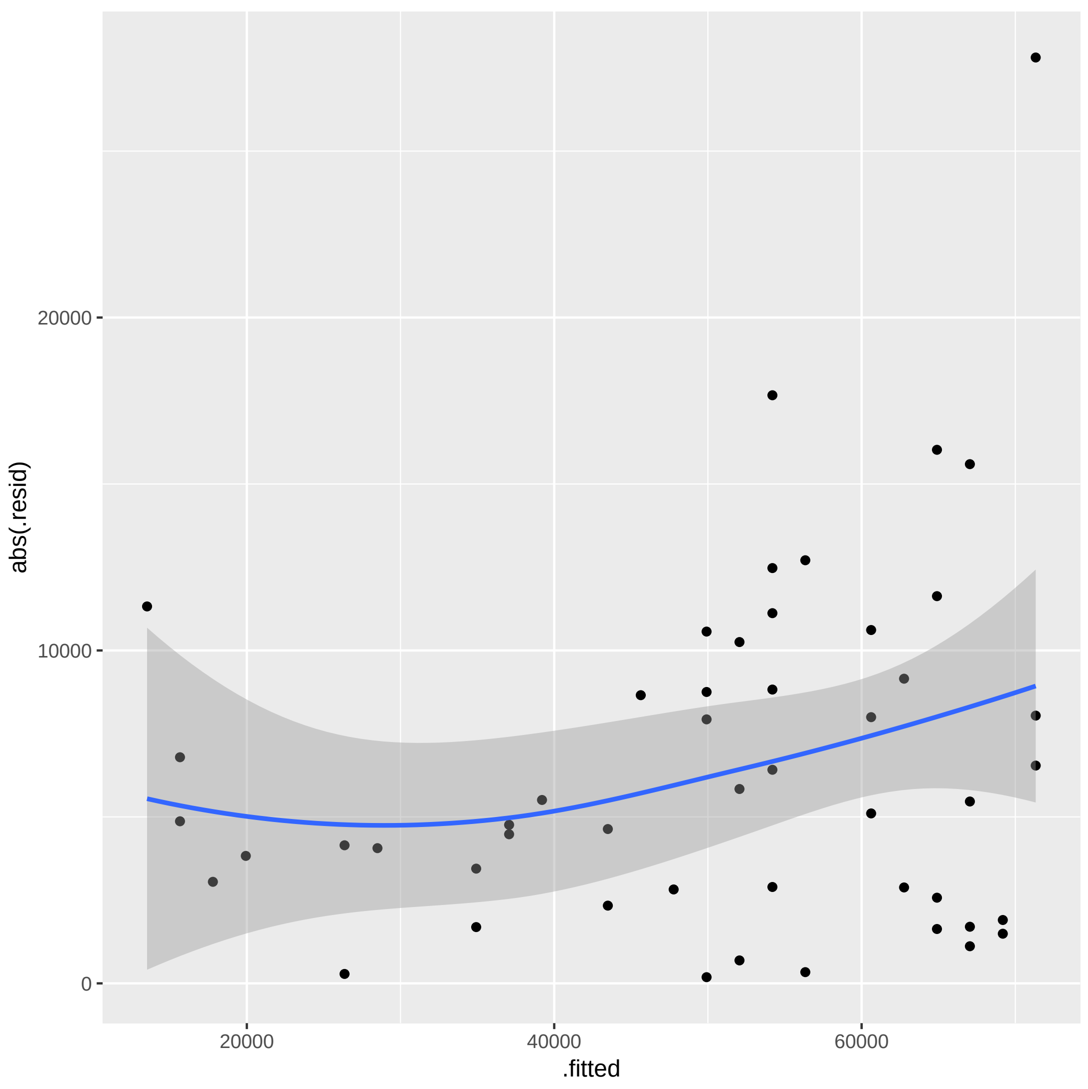
Fanning-out is sometimes hard to see. What you can do if you suspect that it might have happened is to plot the *absolute value* of the residuals against the fitted values. The absolute value is the residual without its plus or minus sign, so if the residuals are getting bigger in size, their absolute 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.

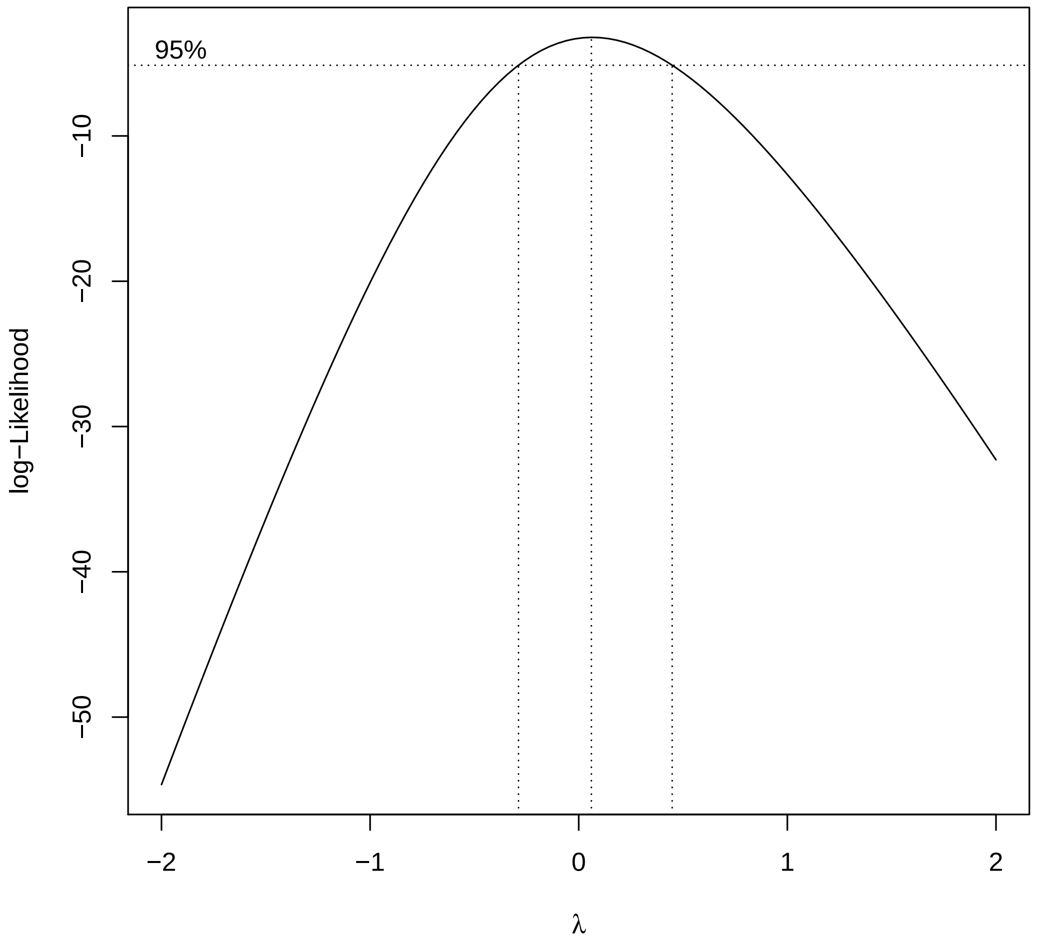
(e) The problem you unearthed in the previous part is often helped by a transformation. Run Box-Cox on your data to find a suitable transformation. What transformation is suggested?

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:[18]

```
search()
```

```
##  [1] ".GlobalEnv"        "package:MASS"      "package:PMCMRplus"
##  [4] "package:carData"   "package:smmr"      "package:bindrcpp"
##  [7] "package:forcats"   "package:stringr"   "package:dplyr"
## [10] "package:purrr"     "package:readr"     "package:tidyr"
## [13] "package:tibble"    "package:ggplot2"   "package:tidyverse"
## [16] "package:stats"     "package:graphics"  "package:grDevices"
## [19] "package:utils"     "package:datasets"  "package:methods"
## [22] "Autoloads"         "package:base"
```

then get rid of `MASS`:

```
detach("package:MASS",unload=T)
```

```
## Warning:  'MASS' namespace cannot be unloaded:
##  namespace 'MASS' is imported by 'PMCMRplus' so cannot be unloaded
```

Now check that it has gone:

```
search()
```

```
##  [1] ".GlobalEnv"        "package:PMCMRplus" "package:carData"
##  [4] "package:smmr"      "package:bindrcpp"  "package:forcats"
##  [7] "package:stringr"   "package:dplyr"     "package:purrr"
## [10] "package:readr"     "package:tidyr"     "package:tibble"
## [13] "package:ggplot2"   "package:tidyverse" "package:stats"
## [16] "package:graphics"  "package:grDevices" "package:utils"
## [19] "package:datasets"  "package:methods"   "Autoloads"
## [22] "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[19] your variables, data frames etc. get stored, and that is what gets searched first.[20] 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.

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 Pr(>|t|)
## (Intercept) 9.841315   0.056356  174.63   <2e-16 ***
## experience  0.049979   0.002868   17.43   <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 Pr(>|t|)
## (Intercept) 9.841315   0.056356  174.63   <2e-16 ***
## experience  0.049979   0.002868   17.43   <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`:

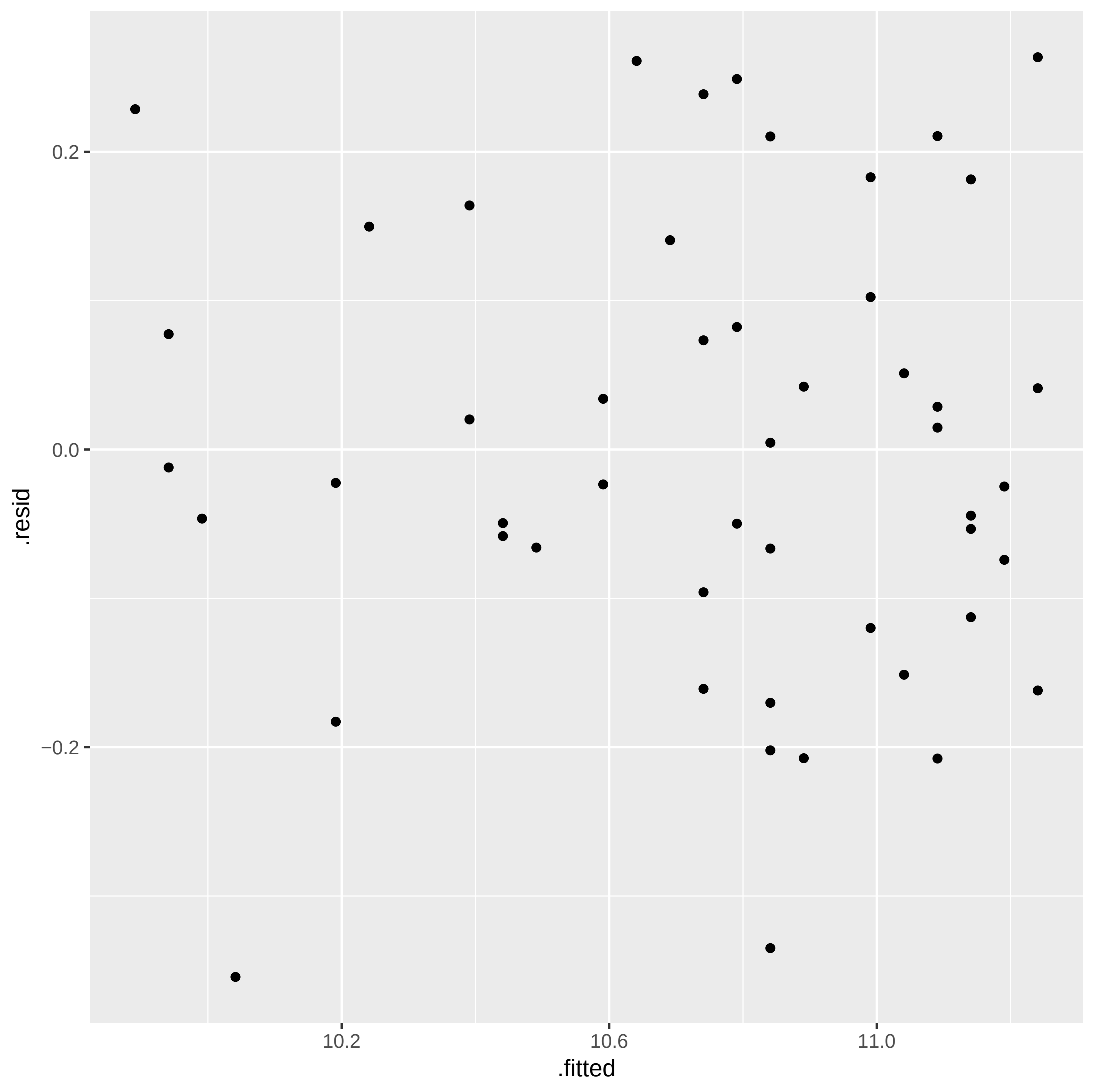
```
soc.1a=lm(log(salary)~experience, data=soc)
summary(soc.1a)
```

```
##
## 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 Pr(>|t|)
## (Intercept) 9.841315   0.056356  174.63   <2e-16 ***
## experience  0.049979   0.002868   17.43   <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:

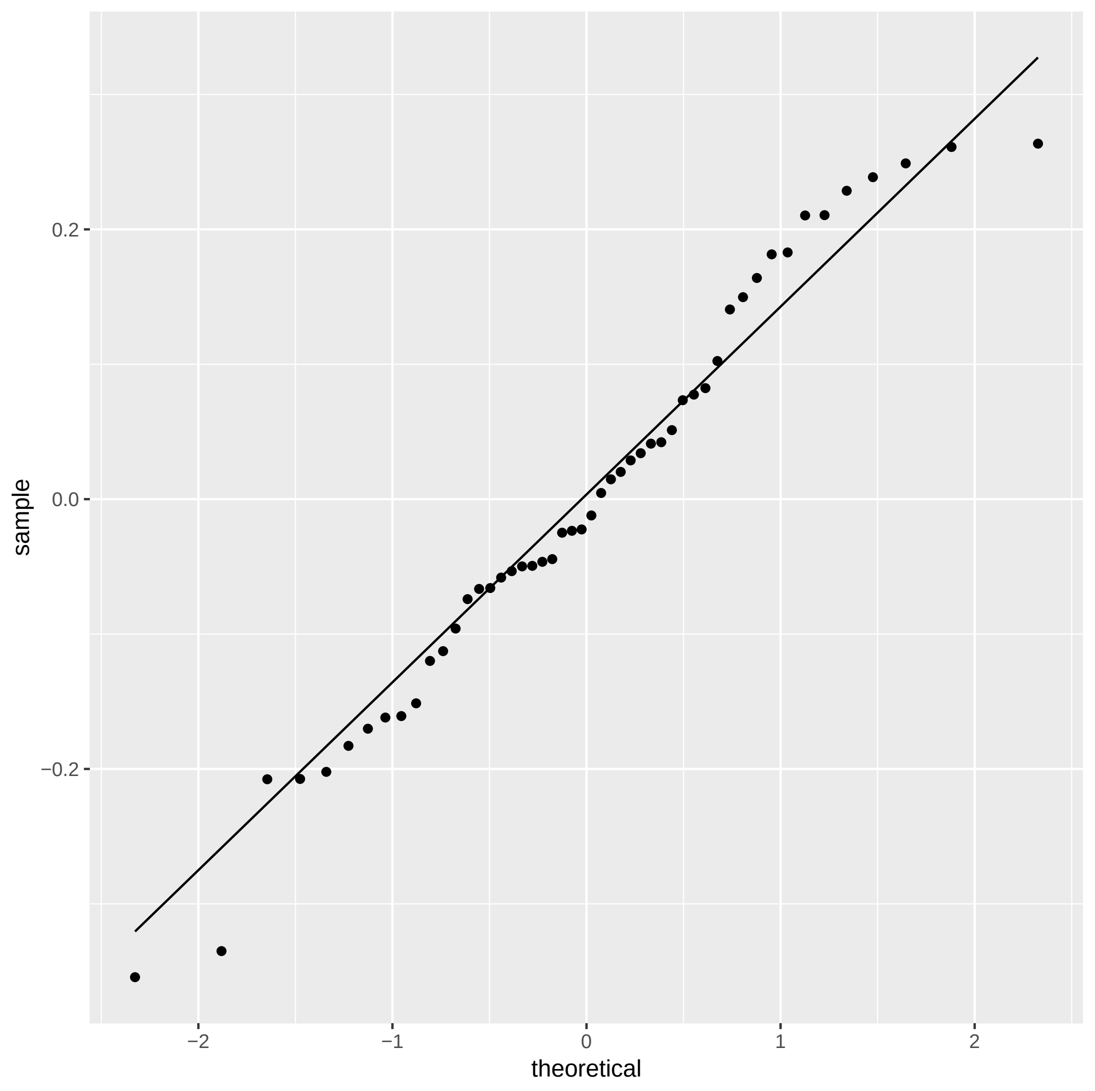
```
ggplot(soc.3,aes(x=.fitted,y=.resid))+geom_point()
```



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%.[21]

Number 1 songs

11. The data file `http://stat405.had.co.nz/data/billboard.csv` 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 date.entered date.peaked
##    <int> <chr>           <chr> <tim> <chr> <date>       <date>
##  1  2000 Destiny's Child Inde~ 03:38 Rock  2000-09-23   2000-11-18
##  2  2000 Santana         Mari~ 04:18 Rock  2000-02-12   2000-04-08
##  3  2000 Savage Garden   I Kn~ 04:07 Rock  1999-10-23   2000-01-29
##  4  2000 Madonna         Music 03:45 Rock  2000-08-12   2000-09-16
##  5  2000 Aguilera, Chri~ Come~ 03:38 Rock  2000-08-05   2000-10-14
##  6  2000 Janet           Does~ 04:17 Rock  2000-06-17   2000-08-26
##  7  2000 Destiny's Child Say ~ 04:31 Rock  1999-12-25   2000-03-18
##  8  2000 Iglesias, Enri~ Be W~ 03:36 Latin 2000-04-01   2000-06-24
##  9  2000 Sisqo           Inco~ 03:52 Rock  2000-06-24   2000-08-12
## 10  2000 Lonestar        Amaz~ 04:25 Coun~ 1999-06-05   2000-03-04
## # ... with 307 more rows, and 76 more variables: 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 date.entered date.peaked week
##   * <int> <chr>           <chr> <tim> <chr> <date>       <date>      <chr>
##   1  2000 Destiny's Child Inde~ 03:38 Rock  2000-09-23   2000-11-18  x1st~
##   2  2000 Santana         Mari~ 04:18 Rock  2000-02-12   2000-04-08  x1st~
##   3  2000 Savage Garden   I Kn~ 04:07 Rock  1999-10-23   2000-01-29  x1st~
##   4  2000 Madonna         Music 03:45 Rock  2000-08-12   2000-09-16  x1st~
##   5  2000 Aguilera, Chri~ Come~ 03:38 Rock  2000-08-05   2000-10-14  x1st~
##   6  2000 Janet           Does~ 04:17 Rock  2000-06-17   2000-08-26  x1st~
##   7  2000 Destiny's Child Say ~ 04:31 Rock  1999-12-25   2000-03-18  x1st~
##   8  2000 Iglesias, Enri~ Be W~ 03:36 Latin 2000-04-01   2000-06-24  x1st~
##   9  2000 Sisqo           Inco~ 03:52 Rock  2000-06-24   2000-08-12  x1st~
## 10  2000 Lonestar        Amaz~ 04:25 Coun~ 1999-06-05   2000-03-04  x1st~
## # ... with 5,297 more rows, and 1 more variable: 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 date.entered date.peaked week
##   * <int> <chr>           <chr> <tim> <chr> <date>       <date>      <chr>
##   1  2000 Destiny's Child Inde~ 03:38 Rock  2000-09-23   2000-11-18  x1st~
##   2  2000 Santana         Mari~ 04:18 Rock  2000-02-12   2000-04-08  x1st~
##   3  2000 Savage Garden   I Kn~ 04:07 Rock  1999-10-23   2000-01-29  x1st~
##   4  2000 Madonna         Music 03:45 Rock  2000-08-12   2000-09-16  x1st~
##   5  2000 Aguilera, Chri~ Come~ 03:38 Rock  2000-08-05   2000-10-14  x1st~
##   6  2000 Janet           Does~ 04:17 Rock  2000-06-17   2000-08-26  x1st~
##   7  2000 Destiny's Child Say ~ 04:31 Rock  1999-12-25   2000-03-18  x1st~
##   8  2000 Iglesias, Enri~ Be W~ 03:36 Latin 2000-04-01   2000-06-24  x1st~
##   9  2000 Sisqo           Inco~ 03:52 Rock  2000-06-24   2000-08-12  x1st~
## 10  2000 Lonestar        Amaz~ 04:25 Coun~ 1999-06-05   2000-03-04  x1st~
## # ... with 5,297 more rows, and 1 more variable: 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, Christina What A Girl Wants     2000-01-15
##  2 Aguilera, Christina What A Girl Wants     2000-01-22
##  3 Savage Garden       I Knew I Loved You    2000-01-29
##  4 Savage Garden       I Knew I Loved You    2000-02-05
##  5 Savage Garden       I Knew I Loved You    2000-02-12
##  6 Carey, Mariah       Thank God I Found You 2000-02-19
##  7 Savage Garden       I Knew I Loved You    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[22] and so is not listed here.

(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 You)     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 You)     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.[23]

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 Destiny's Child                11
##  2 Maria, Maria by Santana                                    10
##  3 Come On Over Baby (All I Want Is You) by Aguilera, Christina     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, Christina                    2
## 13 Bent by matchbox twenty                                     1
## 14 Everything You Want by Vertical Horizon                     1
## 15 Thank God I Found You by Carey, Mariah                      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.)

Predicting volume of wood in pine trees

12. 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 `http://www.utsc.utoronto.ca/~butler/c32/pinetrees.txt`. 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:

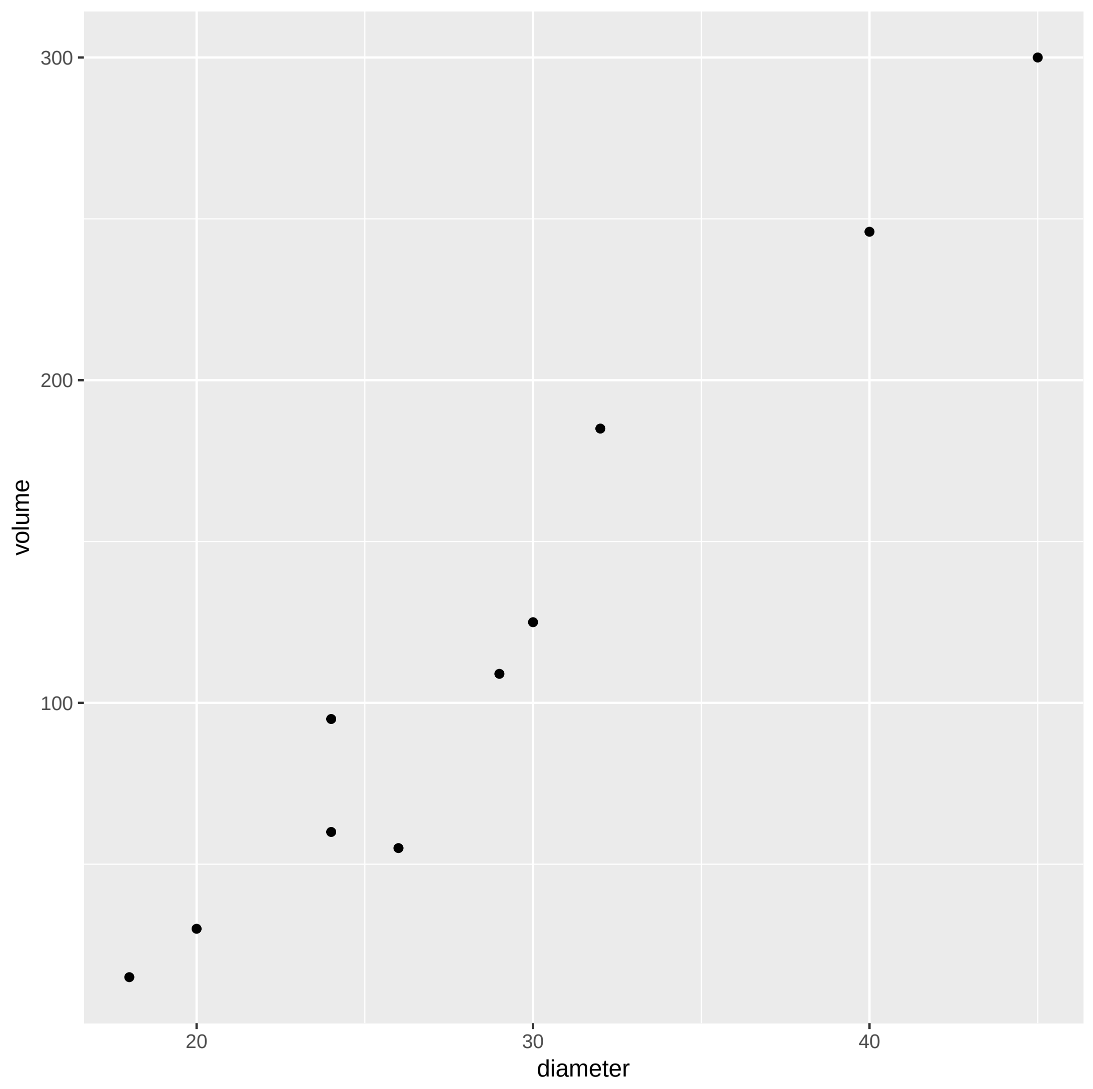
```
my_url="http://www.utsc.utoronto.ca/~butler/c32/pinetrees.txt"
trees=read_delim(my_url," ")

## 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.

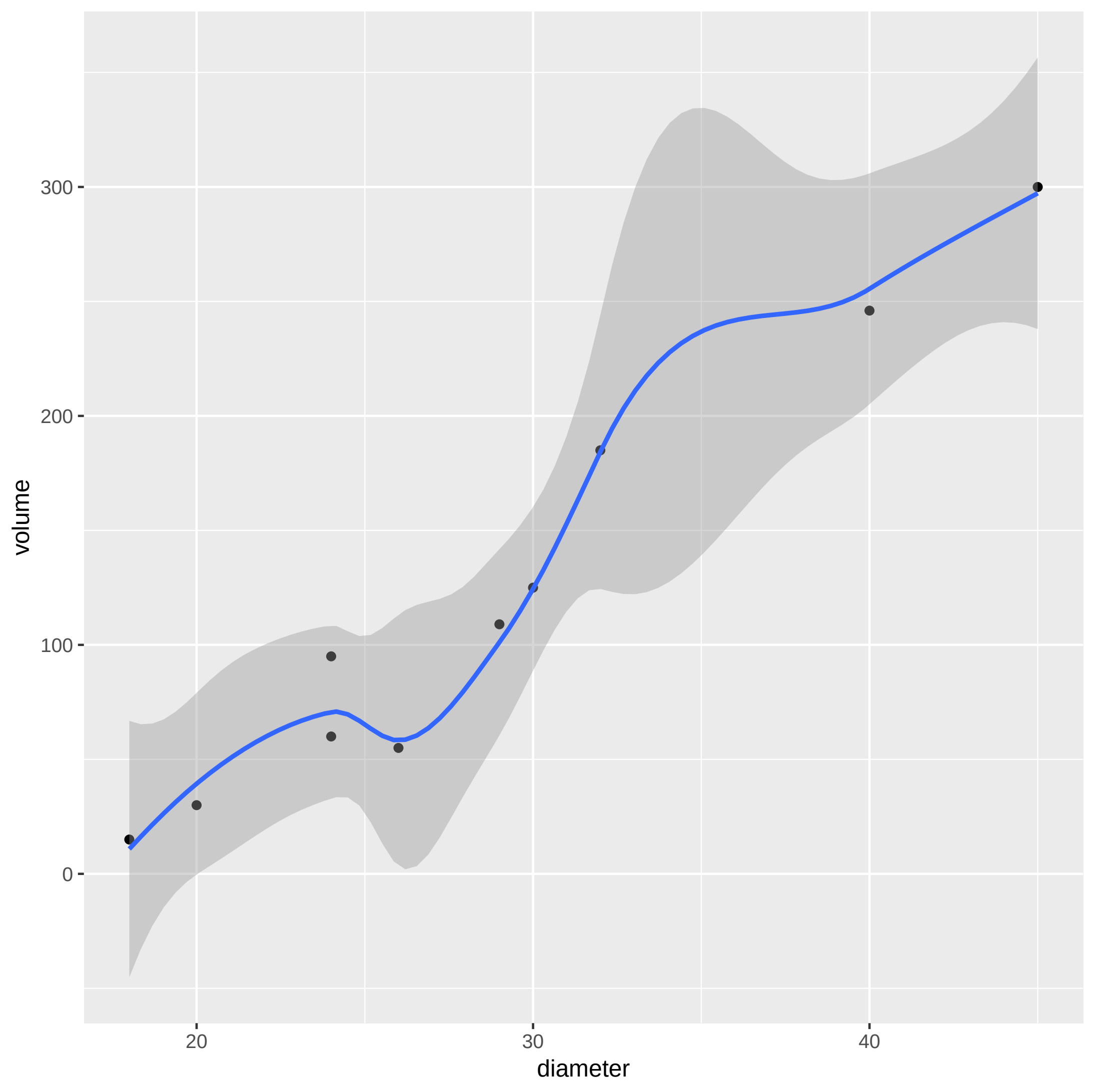
Solution: No clues this time. You need to recognize that you have two quantitative variables, so that a scatterplot is called for. Also, the volume is the response, so that should go on the y-axis:

```
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.[24]

(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 Pr(>|t|)
## (Intercept) -191.749     23.954  -8.005 4.35e-05 ***
## diameter      10.894      0.801  13.600 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 p.value    df logLik   AIC   BIC
## *     <dbl>         <dbl> <dbl>     <dbl>   <dbl> <int>  <dbl> <dbl> <dbl>
## 1     0.959         0.953  20.4      185. 8.22e-7     2  -43.2  92.4  93.4
## # ... with 2 more variables: 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 (Intercept)   -192.      24.0       -8.01 0.0000435
## 2 diameter        10.9      0.801     13.6  0.000000822
```

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:[25]

```
tidy(volume.1) %>% filter(term=="diameter")
```

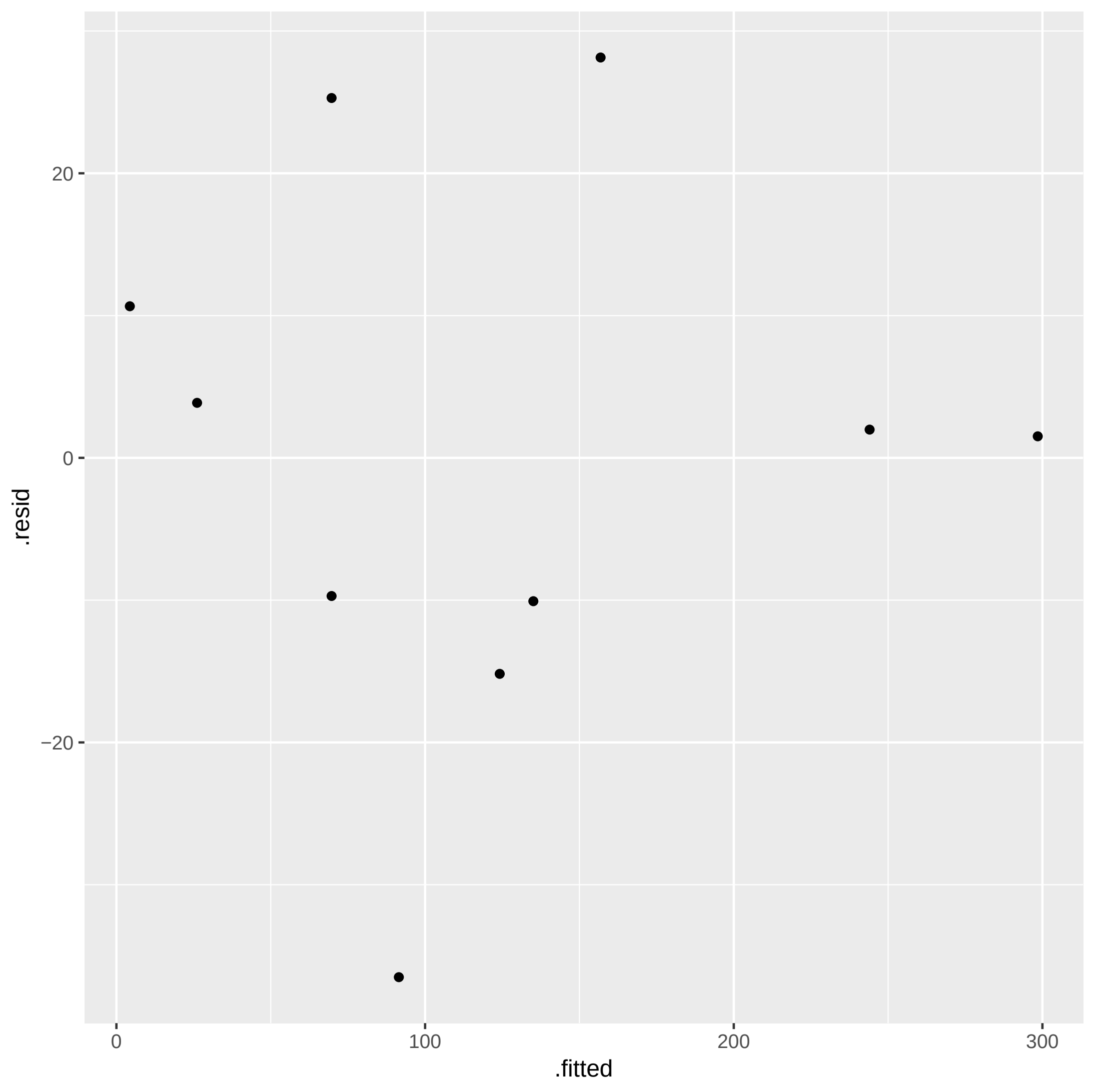
```
## # A tibble: 1 x 5
##   term     estimate std.error statistic     p.value
##   <chr>       <dbl>     <dbl>     <dbl>       <dbl>
## 1 diameter     10.9     0.801      13.6 0.000000822
```

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.

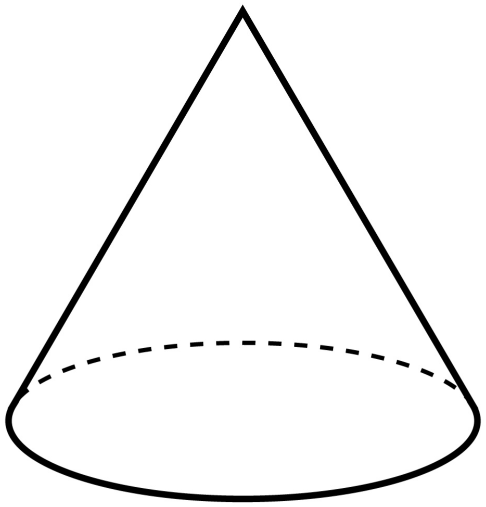
My feeling is that the residuals on the right are close to 0 because these points have noticeably larger diameter than the others, and they are *influential* points in the regression that will pull the line closer to themselves. This is why their residuals are close to zero. But I am happy with either 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 `http://www.web-formulas.com/Math_Formulas/Geometry_Volume_of_Cone.aspx`, 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)
```

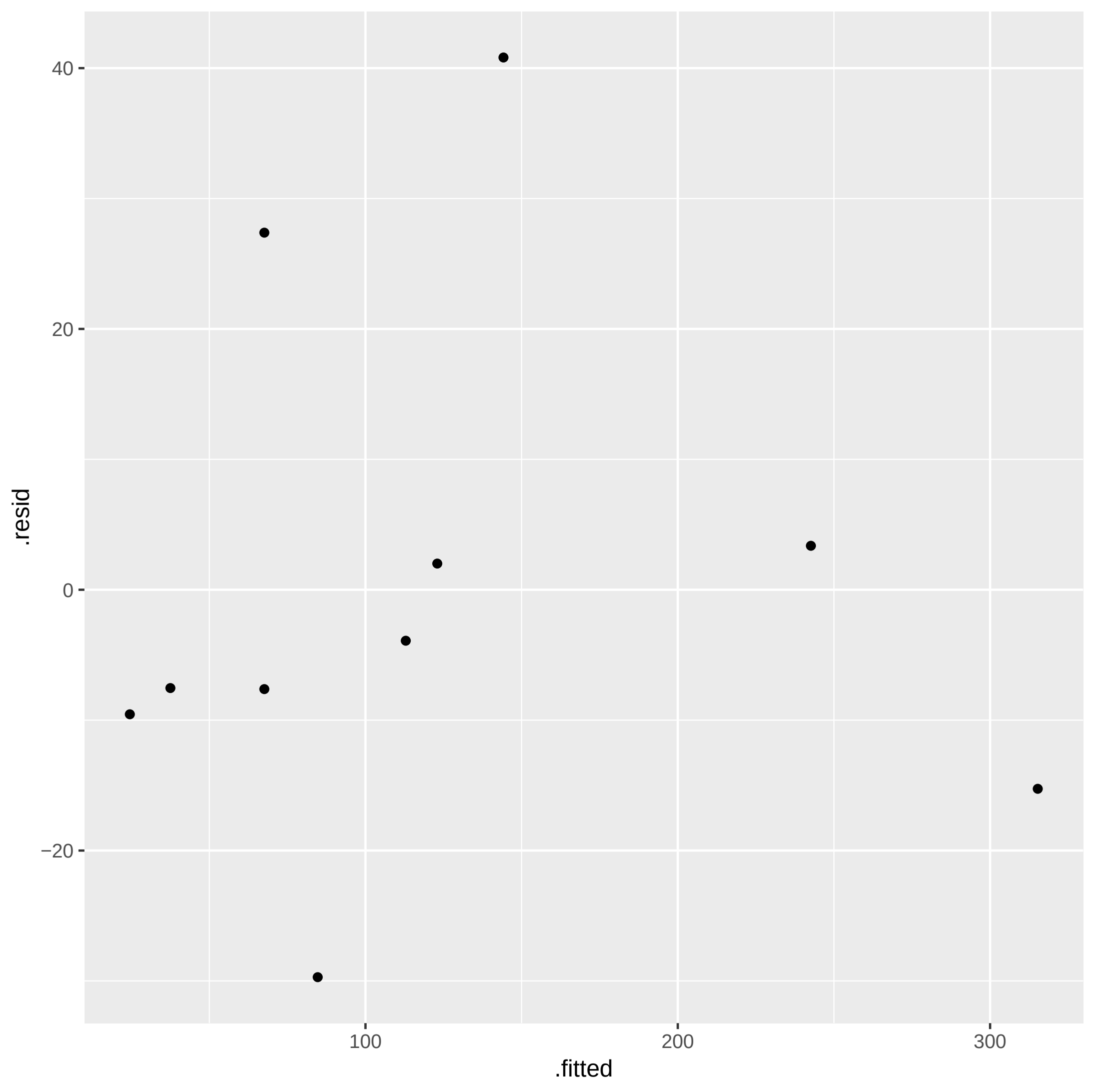
```
##
## Call:
## lm(formula = volume ~ I(diameter^2), data = trees)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -29.708  -9.065  -5.722   3.032  40.816
##
## Coefficients:
##                Estimate Std. Error t value Pr(>|t|)
## (Intercept)   -30.82634   13.82243   -2.23   0.0563 .
## I(diameter^2)   0.17091    0.01342   12.74 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: `https://pdfs.semanticscholar.org/5497/3d02d63428e3dfed6645acfdba874ad80822.pdf`. 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 Pr(>|t|)
## (Intercept)    -5.9243     1.1759  -5.038    0.001 **
## log(diameter)   3.1284     0.3527   8.870 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:

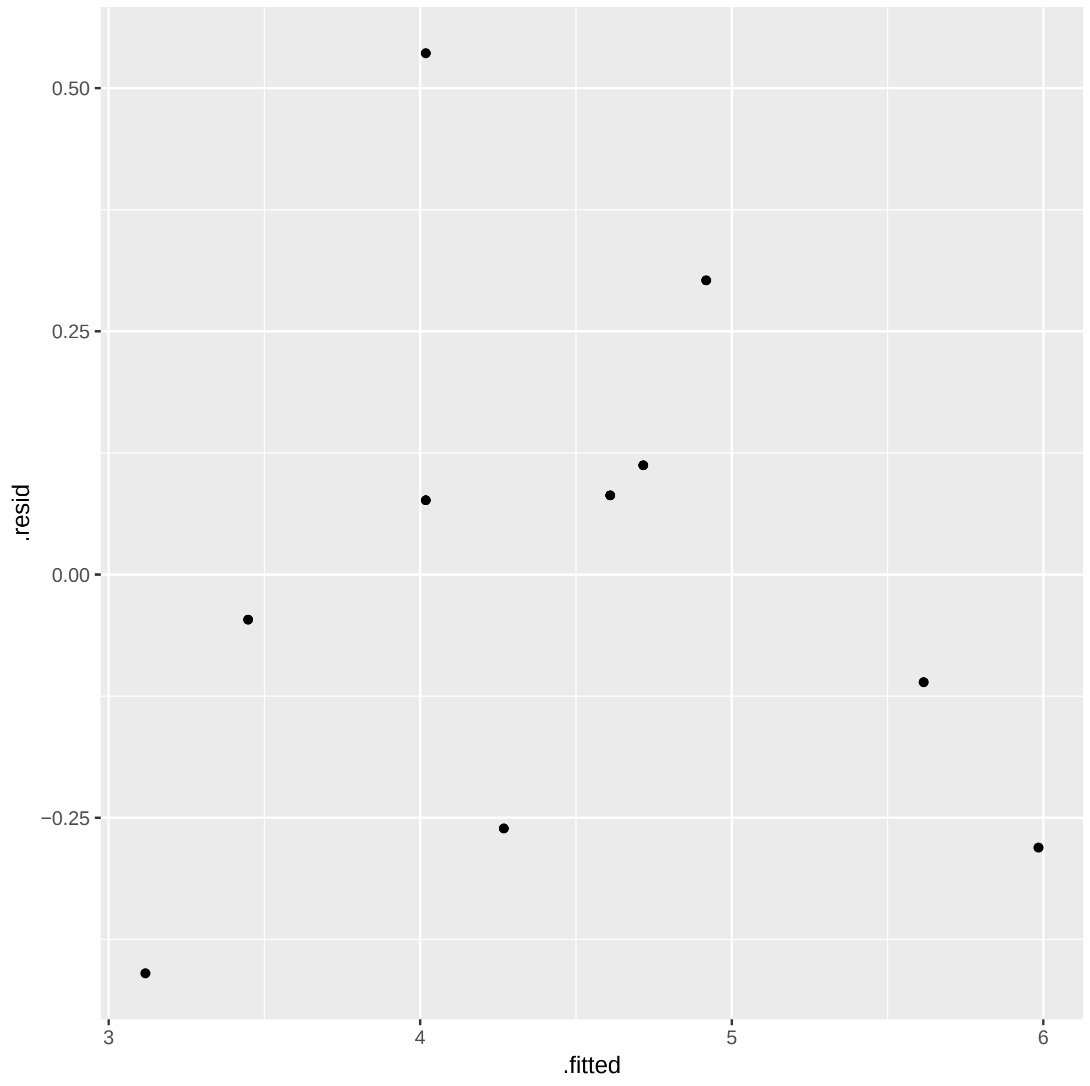
```
confint(volume.3)
```

```
##                   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.

¡br¿

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 p.value    df logLik   AIC   BIC
##       <dbl>         <dbl>  <dbl>     <dbl>   <dbl> <int>  <dbl> <dbl> <dbl>
## 1     0.959         0.953 20.4       185.  8.22e-7     2 -43.2  92.4  93.4
## 2     0.953         0.947 21.7       162.  1.36e-6     2 -43.8  93.7  94.6
## 3     0.908         0.896  0.303      78.7 2.06e-5     2  -1.12  8.25  9.16
## # ... with 2 more variables: 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 p.value    df logLik   AIC   BIC
##       <dbl>         <dbl>  <dbl>     <dbl>   <dbl> <int>  <dbl> <dbl> <dbl>
## 1     0.959         0.953 20.4       185.  8.22e-7     2 -43.2  92.4  93.4
## 2     0.953         0.947 21.7       162.  1.36e-6     2 -43.8  93.7  94.6
## 3     0.908         0.896  0.303      78.7 2.06e-5     2  -1.12  8.25  9.16
## # ... with 2 more variables: 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,[26] 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).

Hand the next two questions in:

13. A fast-food chain that specializes in Mexican food (tacos, burritos etc.) is opening a new franchise in a university town. An important consideration in determining where the franchise will be located is traffic density. Five possible locations, labelled I, II, III, IV and V, are being considered. At each one, the number of cars passing by each location was counted on each of 10 different days. Two of the observations were missing (this will not affect our analysis).

    The data are at `https://www.utsc.utoronto.ca/~butler/c32/texmex.txt`.

    (a) (2 marks) Read in and display the data, confirming that you have what was described.

Solution: Separated by single spaces:

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

```
## Parsed with column specification:
## cols(
##   I = col_integer(),
##   II = col_integer(),
##   III = col_integer(),
##   IV = col_integer(),
##   V = col_integer()
## )
```

```
texmex
```

```
## # A tibble: 10 x 5
##         I    II   III    IV     V
##     <int> <int> <int> <int> <int>
##   1    44   412   237   518   367
##   2   382   441   390   501   445
##   3   353   607   365   577   480
##   4   395   531   355   642   323
##   5   207   486   217   489   366
##   6   312   508   268   475   325
##   7   407   337   117   532   316
##   8   421   419   273   540   381
##   9   366   499   288    NA   407
##  10   222   387   351    NA   339
```

10 rows (days), 5 locations (columns). You can even note the two missing values.

Extra: we are assuming independent observations, to validate the ANOVA we'll be doing later. This means that we had 50 different days (well, 48), which you might imagine if there is a traffic-counting crew that can only do one location on one day. In that case, the particular day that each location gets should have been randomly chosen. Or, there were only 10 different days, in which case we strictly have repeated measures, but we might be able to make the case that traffic density at different locations on the same day is independent (for example, if you can say that having heavy traffic at location I on Monday says nothing about how the traffic will be at the other locations on the same day.)

As you will see, the analysis itself is not that hard, but getting to the point where you can justify doing that analysis involves a lot more thinking.

(b) (2 marks) Explain briefly how your data are not tidy (bearing in mind that we'll be doing an analysis of variance later).

Solution: An analysis of variance needs one quantitative and one categorical variable, but here we have five columns. The location is spread over five columns, and all the numbers are traffic density, so they should be in one column labelled by location.

You need to get at least some reasonable fraction of this. The hint about ANOVA later was to guide your thinking about what you want, and possibly about how you might get it.

(c) (3 marks) Produce a tidy data frame that is ready for analysis. Show at least the first few rows of your tidy data frame.

Solution: This is a classic `gather`, since you want to gather up those five columns into one. What makes the columns different is that they are different locations; what makes them the same is that they are all measures of the amount of traffic, so:

```
(texmex %>% gather(location, traffic, everything()) -> texmex2)

## # A tibble: 50 x 2
##    location traffic
##    <chr>      <int>
##  1 I             44
##  2 I            382
##  3 I            353
##  4 I            395
##  5 I            207
##  6 I            312
##  7 I            407
##  8 I            421
##  9 I            366
## 10 I            222
## # ... with 40 more rows
```

The ones displayed are all the location I measurements; on yours, you can click Next to display location II and so on. Feel free to use `I:V` to select the columns to gather instead of `everything()`.

This is ready for `aov`.

Extra: I think `gather` will remove missings, optionally:

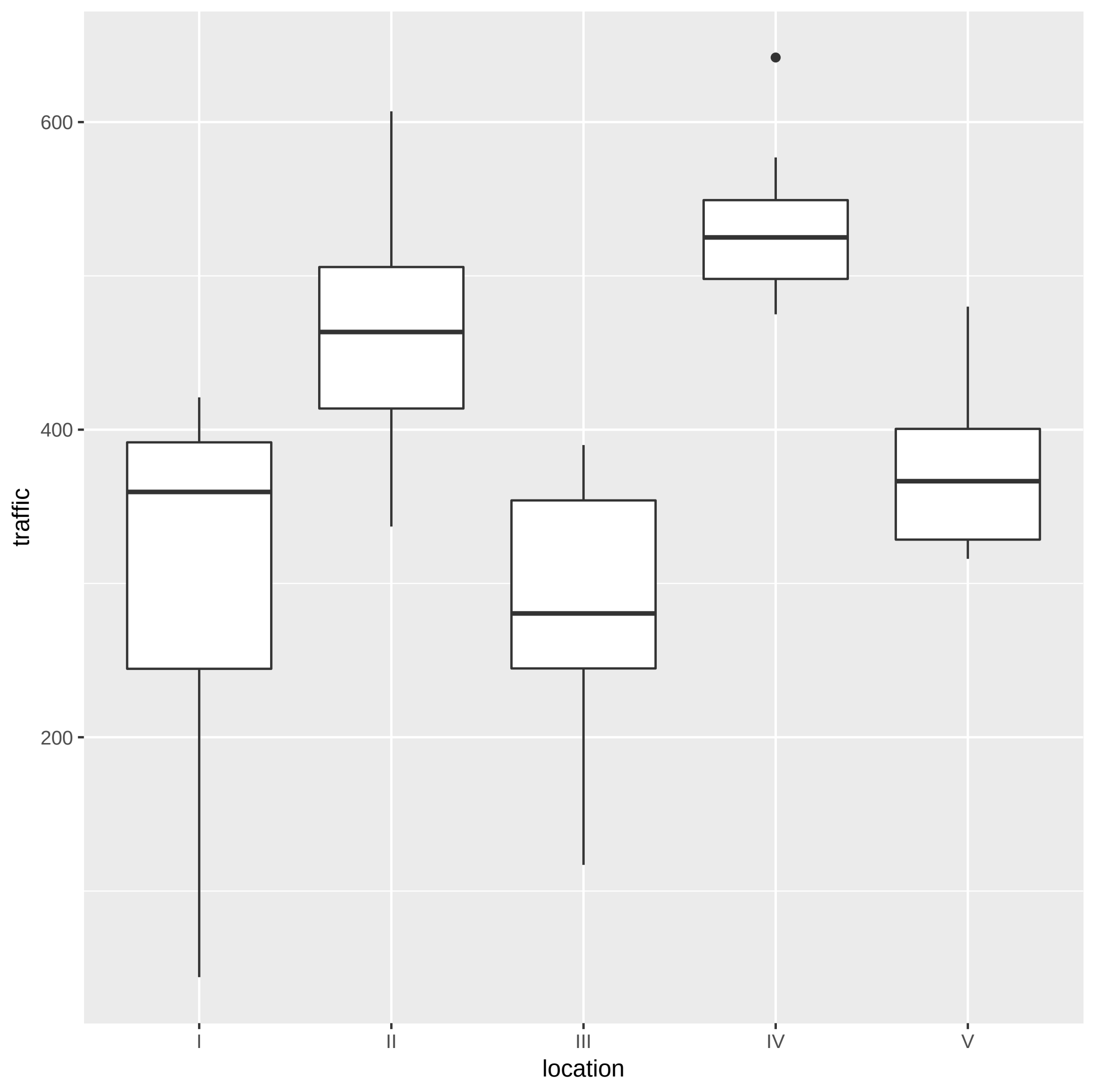
```
texmex %>% gather(location, traffic, everything(), na.rm=T) -> texmex3
texmex3 %>% count(location)

## # A tibble: 5 x 2
##   location     n
##   <chr>    <int>
## 1 I           10
## 2 II          10
## 3 III         10
## 4 IV           8
## 5 V           10
```

They're gone. (I saved `texmex3` because I'll be using it later.)

(d) (2 marks) Make a suitable plot of your tidy data frame. (Depending on how you did the tidying, you might get a warning about "non-finite values", which you can ignore.)

Solution: One quantitative, one categorical suggests boxplots. The warning messages come from the missing values. (If I had used the data frame with `na.rm=T`, I would not have gotten the warnings.)

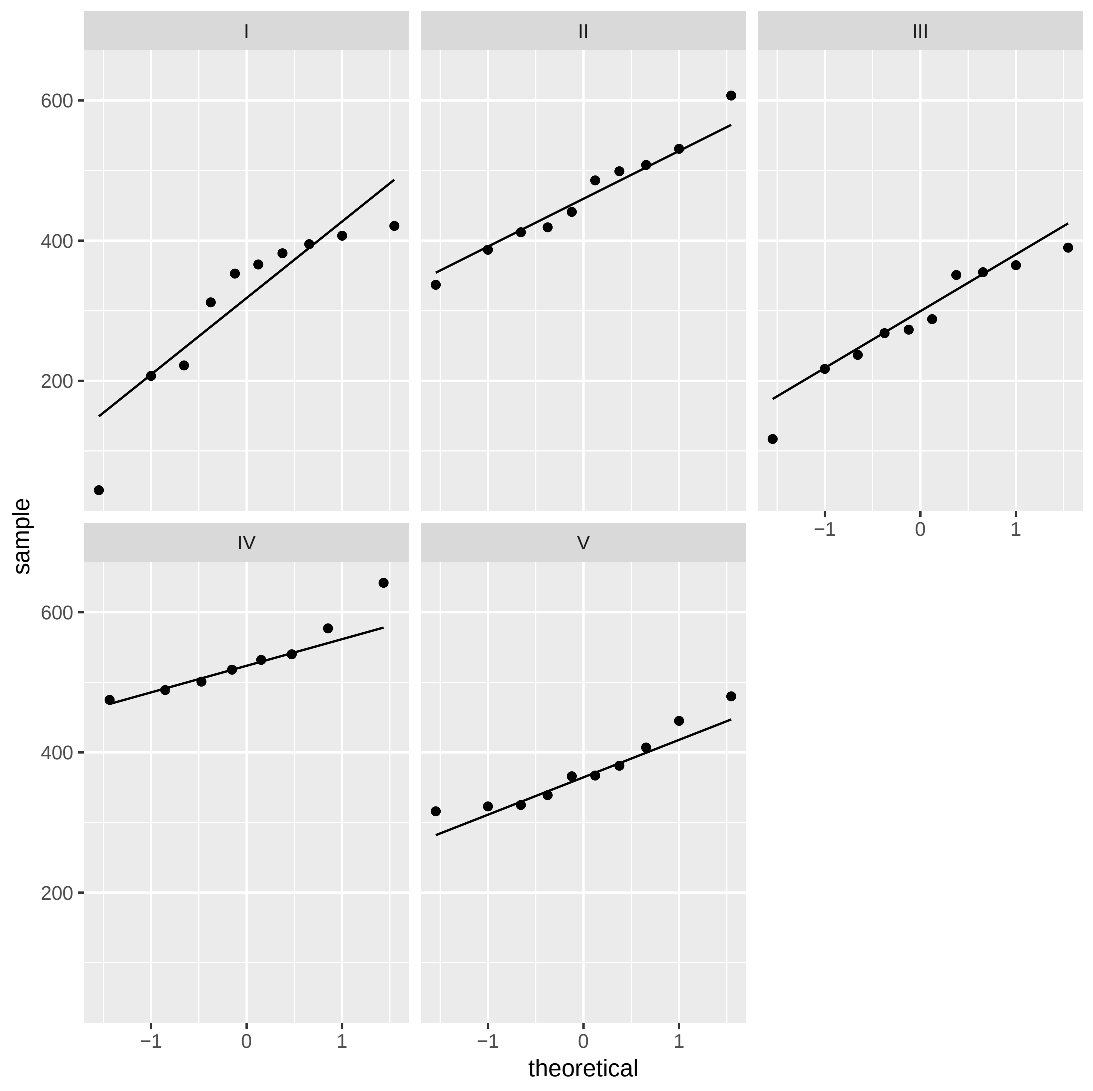
```
ggplot(texmex2, aes(x=location, y=traffic))+geom_boxplot()

## Warning:  Removed 2 rows containing non-finite values (stat_boxplot).
```



I didn't ask you for comment (that's coming later). If I had, you would have assessed these for approximate normality and equal spreads.

Or, if you like (bearing in mind our purposes), make a normal quantile plot for each location:

```
ggplot(texmex2, aes(sample=traffic))+
    stat_qq()+stat_qq_line()+
    facet_wrap(~location)

## Warning:  Removed 2 rows containing non-finite values (stat_qq).

## Warning:  Removed 2 rows containing non-finite values (stat_qq_line).
```



Once again, the warnings are because of the missing values.

The one thing a normal quantile plot doesn't (obviously) give you is spread, which you could assess by looking at the *slopes* of the lines: location I has the most spread and location IV has the least.

(e) (3 marks) Run an analysis of variance, bearing in mind that we might be running Tukey shortly. (You may assume that the assumptions for analysis of variance are satisfied here.) What do you conclude?

Solution: `aov`, therefore, using your tidy data frame:

```
texmex.1=aov(traffic~location,data=texmex2)
summary(texmex.1)

##             Df Sum Sq Mean Sq F value   Pr(>F)
## location     4 391909   97977   14.34 1.63e-07 ***
## Residuals   43 293888    6835
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
## 2 observations deleted due to missingness
```

The P-value of 1.63×10^{-7} is really small, so the locations are not all the same in terms of mean traffic, or one or more of the locations have different mean traffic than the others, or equivalent.

Don't say anything about *which* locations are different, yet. That's coming up!

Extra: You might also reasonably have looked at your boxplots and said "that's not normal enough", in which case you should run something like Mood's median test. My implementation of Mood's median test doesn't handle missing values, so you need to use a tidy data frame with the missings removed, such as the one I called `texmex3`:

```
median_test(texmex3, traffic, location)

## $table
##      above
## group above below
##   I       3     7
##   II      9     1
##   III     1     9
##   IV      8     0
##   V       3     7
##
## $test
##        what        value
## 1 statistic 2.400000e+01
## 2        df 4.000000e+00
## 3   P-value 7.987476e-05
```

The P-value is not quite as small as the ANOVA, but the conclusion is the same thing: the five locations do not all have the same median traffic.

(f) (2 marks) Run Tukey's method, if appropriate. (If not appropriate, explain briefly why not.)

Solution: There are significant differences to find, and Tukey's method will help us find them. So on we go, feeding the `aov` output into `TukeyHSD`:

```
TukeyHSD(texmex.1)
```

```
##   Tukey multiple comparisons of means
##     95% family-wise confidence level
##
## Fit: aov(formula = traffic ~ location, data = texmex2)
##
## $location
##            diff        lwr       upr     p adj
## II-I     151.80   46.54516 257.05484 0.0015822
## III-I    -24.80 -130.05484  80.45484 0.9616384
## IV-I     223.35  111.71039 334.98961 0.0000097
## V-I       64.00  -41.25484 169.25484 0.4263740
## III-II  -176.60 -281.85484 -71.34516 0.0001957
## IV-II     71.55  -40.08961 183.18961 0.3731687
## V-II     -87.80 -193.05484  17.45484 0.1419592
## IV-III   248.15  136.51039 359.78961 0.0000012
## V-III     88.80  -16.45484 194.05484 0.1343769
## V-IV    -159.35 -270.98961 -47.71039 0.0017972
```

With five groups, this is complicated to interpret, so I'm not going to ask you to.

This was what I originally intended, with the reason you would *not* run Tukey being that the ANOVA F-test was not significant, if that had been the case.

If you decided to run Mood's median test, here you should be consistent and run the pairwise median tests (and explain why it was that you were not running Tukey: you didn't feel the data were normal enough):

```
pairwise_median_test(texmex3, traffic, location)
```

```
## # A tibble: 10 x 4
##    g1    g2     p_value adj_p_value
##    <chr> <chr>    <dbl>       <dbl>
##  1 I     II    0.00729      0.0729
##  2 I     III   0.371        3.71
##  3 I     IV    0.000148     0.00148
##  4 I     V     0.637        6.37
##  5 II    III   0.000347     0.00347
##  6 II    IV    0.0578       0.578
##  7 II    V     0.00729      0.0729
##  8 III   IV    0.000148     0.00148
##  9 III   V     0.371        3.71
## 10 IV    V     0.000148     0.00148
```

Once again, don't worry about interpreting this yet.

(g) (3 marks) What location or locations would you recommend for the franchise? Justify your answer briefly.

Solution: The franchise is looking for a location with a lot of traffic (that they would expect to translate into a lot of sales). Location IV has the highest median traffic, according to the boxplot. But is it significantly higher than the other locations? Look at the Tukey output.

Location IV is significantly higher in terms of traffic than all the other locations *except for location II*. So our recommendation should be either location IV or location II. The ANOVA does not enable us to distinguish between them; for example, if the traffic surveys had been done on different days, location II could have come out higher (that's what "not significantly different" means). So you need to recommend both of these.

Another way to do this is to work out the mean traffic for each location *first*. Don't forget to use a *tidy* data frame! You'll also need to make sure that the calculation of the means is not messed up by the missing values (if they are still around):

```
texmex2 %>% group_by(location) %>%
    summarize(m=mean(traffic, na.rm=T)) %>%
    arrange(desc(m))
```

```
## # A tibble: 5 x 2
##   location     m
##   <chr>    <dbl>
## 1 IV        534.
## 2 II        463.
## 3 V         375.
## 4 I         311.
## 5 III       286.
```

Start at the top of the list (I arranged them into order). Location IV has the most traffic on average. This location is *not* significantly different from location II (check the Tukey), but it is significantly different from all the other locations. Thus we'd recommend either location IV (best) or location II (equivalent-to-best).

When I learned to do this by hand many[27] years ago, we would do Tukey by writing the means out in order, and then joining by lines the ones that are not significantly different:[28]

```
Location      III     I     V    II   IV
Mean traffic  286   311   375   463  534
              ---------------
                        ---------
                              --------
```

This shows, perhaps more clearly, that location IV is best but not significantly different from location II. (Location II is in a kind of logical grey zone: not significantly worse than IV, but not significantly better than V, even though IV and V *are* significantly different from each other. This happens in ANOVA because of the way hypothesis testing works. II might be worse than IV but we couldn't prove it.)

If you did the pairwise median tests, interpret the results in the same way. The highest median (from the boxplot) was for location IV; this was significantly different from all the locations except location II, so the recommendation is the same as from the Tukey: either location IV or location II (because we don't have the evidence to choose between them).

Whichever way you went, test-wise, you ended up with the same conclusion, so the upshot is that it didn't really matter which way you went test-wise (or, said another way, the normality assumption might have been rather shaky, but it wasn't shaky enough to change the conclusions).

Extra: I decided to stop the question here, because the point of the question as written was to get you to do `gather`, `aov` and Tukey. In practice, though, you would assess your plots for approximate normality and equal spreads before proceeding. There is an outlier at location IV, and some left skewness, especially at location I (maybe right-skewness at location V as well), but maybe we should not be too picky given that there are only 10 observations in each group. If you look at my normal quantile plots, location I is most obviously skewed, and you might say that the others are not too bad (or you might not, up to you.) My take is that you could reasonably criticize doing the ANOVA here, but as with other examples we've seen, the results are so clear that our assumptions could be off by a bit without affecting them.

As is often the case with this kind of question, there is a tactical issue on my part in that I want to get you practicing the techniques, but a lot of the data you might run into has potential issues with it. So in practice you'd assess the assumptions first, but in assessment questions I might have you do it at the end (or not at all), so as to persuade you to run the ANOVA. Here, though, I'm not going to quibble if you ran Mood's median test, *as long* as you follow it up with the pairwise median tests rather then Tukey. Be consistent, above all. (Running Mood's median test followed by Tukey is an *error*, because it is inconsistent, so be prepared to lose something if you do this. At the very least, if you ran Mood's median test, you should *refuse* to run Tukey on the grounds that this would be inconsistent with what you did before.)

14. The Boston Marathon is one of the oldest distance running races. It has been held every year since 1897. There are now divisions for men, women and male and female wheelchair athletes. Our interest is in whether winning times have gotten faster over the years, and if so, how. The data for this question are at `https://www.utsc.utoronto.ca/~butler/c32/marathon.csv`.

The aim of this question is for you to write a report describing your findings, and to make it read like a report *you* came up with, rather than a list of questions I set you to do. With that in mind, you need to supply some narrative text to explain what you are doing and why. The grader will have some discretionary marks to award for the quality of your writing. (This is practice for the data analysis project that you will be working on later.) The grader has three discretionary marks to award for your writing overall, thus:

- 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) (2 marks) Write an introduction that expresses, *in your own words*, what the data are and what you hope to find out from them. (If you copy my words, expect to have the grader note this and give you *zero* for this part.)

Solution: You'll notice that my assignment (and exam) questions have a "preamble" that tells you what the data is about (and, sometimes, what we are trying to discover). This can be the raw material for your Introduction. In it, you can say what the Marathon race is, that one has been held annually in Boston for a long time and that it is one of the most prestigious of the marathons on the world circuit. (The data for this question came from the Wikipedia page `https://en.wikipedia.org/wiki/List_of_winners_of_the_Boston_Marathon`, which links to `https://en.wikipedia.org/wiki/Boston_Marathon`. The second of those provides a lot of background.)

As for our objectives, we are looking to investigate how winning times have changed over the years, or whether runners are faster than in the past, something like that. (If you like, read the rest of the question first to get a sense of what we're trying to do, and then come back and write this. As a general principle, it can be easier to write the Introduction at the end, once you have a better idea of where the analysis is going.)

(b) (3 marks) Read in and display the data. Do you have sensible columns and a sensible number of rows? What is the units of the winning time?

Solution: Reading in first. This is a `.csv`, so nothing special:

```
my_url="https://www.utsc.utoronto.ca/~butler/c32/marathon.csv"
marathon=read_csv(my_url)
```

```
## Parsed with column specification:
## cols(
##   Gender = col_character(),
##   Year = col_integer(),
##   WinTime = col_double()
## )
```

```
marathon
```

```
## # A tibble: 176 x 3
##    Gender  Year WinTime
##    <chr>  <int>   <dbl>
##  1 Men     1897    2.92
##  2 Men     1898    2.7
##  3 Men     1899    2.91
##  4 Men     1900    2.66
##  5 Men     1901    2.49
##  6 Men     1902    2.72
##  7 Men     1903    2.69
##  8 Men     1904    2.63
##  9 Men     1905    2.64
## 10 Men     1906    2.76
## # ... with 166 more rows
```

There are three columns: a column `Gender` that expresses whether the marathon referred to was for men or women; the `Year`; and the winning time, `WinTime`, in hours. (You might know that a good marathon runner takes a bit over 2 hours to complete the race, or you can go back to the Wikipedia page with the winning times and see that the times appear to be hours, minutes and seconds.) Eyeballing the first one, it's just under 3 hours, so the figure of 2.92 hours in the data set looks about right.

I turned the times into decimal hours to make life easier for you later.

If you didn't see that the times on the Wikipedia page were hours, minutes and seconds, you can reason out that these must be decimal hours by finding out how long a marathon race is (42 km) and guessing that these elite runners might be able to run about 20 km/h, which would make the race take a bit over 2 hours. (Back in my running days, I ran a 10k in about 45 minutes, which would probably have scaled to about 5 hours for a marathon, if I'd been able to keep going that long.)

As for the number of rows: the men have been racing for over 100 years, and the women for about 50 ("since the 1960s"), so having a bit over $100 + 50 = 150$ rows looks about right.

All of this is sanity-checking the data to make sure we have something believable.

(c) (2 marks) Women have had a separate marathon only since the 1960s, but men have raced since the beginning. Do your data support this? Show how, obtaining some suitable output.

Solution: The last stage of this is to find the first year of the men's and women's marathon races. Displaying the data shows that the first year of the men's race was 1897 (as I said); to find the first year of the women's race, you can display just the women's times:

```
marathon %>% filter(Gender=="Women")

## # A tibble: 54 x 3
##    Gender  Year WinTime
##    <chr>  <int>   <dbl>
##  1 Women   1966    3.36
##  2 Women   1967    3.45
##  3 Women   1968    3.5
##  4 Women   1969    3.38
##  5 Women   1970    3.09
##  6 Women   1971    3.14
##  7 Women   1972    3.17
##  8 Women   1973    3.10
##  9 Women   1974    2.79
## 10 Women   1975    2.71
## # ... with 44 more rows
```

or you can do both by displaying the minimum year for each Gender, which is a group-by and summarize:

```
marathon %>%
    group_by(Gender) %>%
    summarize(first=min(Year))

## # A tibble: 2 x 2
##   Gender first
##   <chr>  <dbl>
## 1 Men     1897
## 2 Women   1966
```

The first men's race was the first race, in 1897; the first women's race was in 1966. (For a few years prior to that, women were allowed to compete in the men's race.)

There are undoubtedly other ways to find out how long the men and women have had races in the Boston Marathon. Find one of them, and I'm good. It is, to my mind, reasonable to assume that once a race starts, it has continued to the present day, but you can check that as well if you like:

```
marathon %>%
    group_by(Gender) %>%
    summarize(first=min(Year), last=max(Year))
```

```
## # A tibble: 2 x 3
##   Gender first  last
##   <chr>  <dbl> <dbl>
## 1 Men     1897  2018
## 2 Women   1966  2018
```

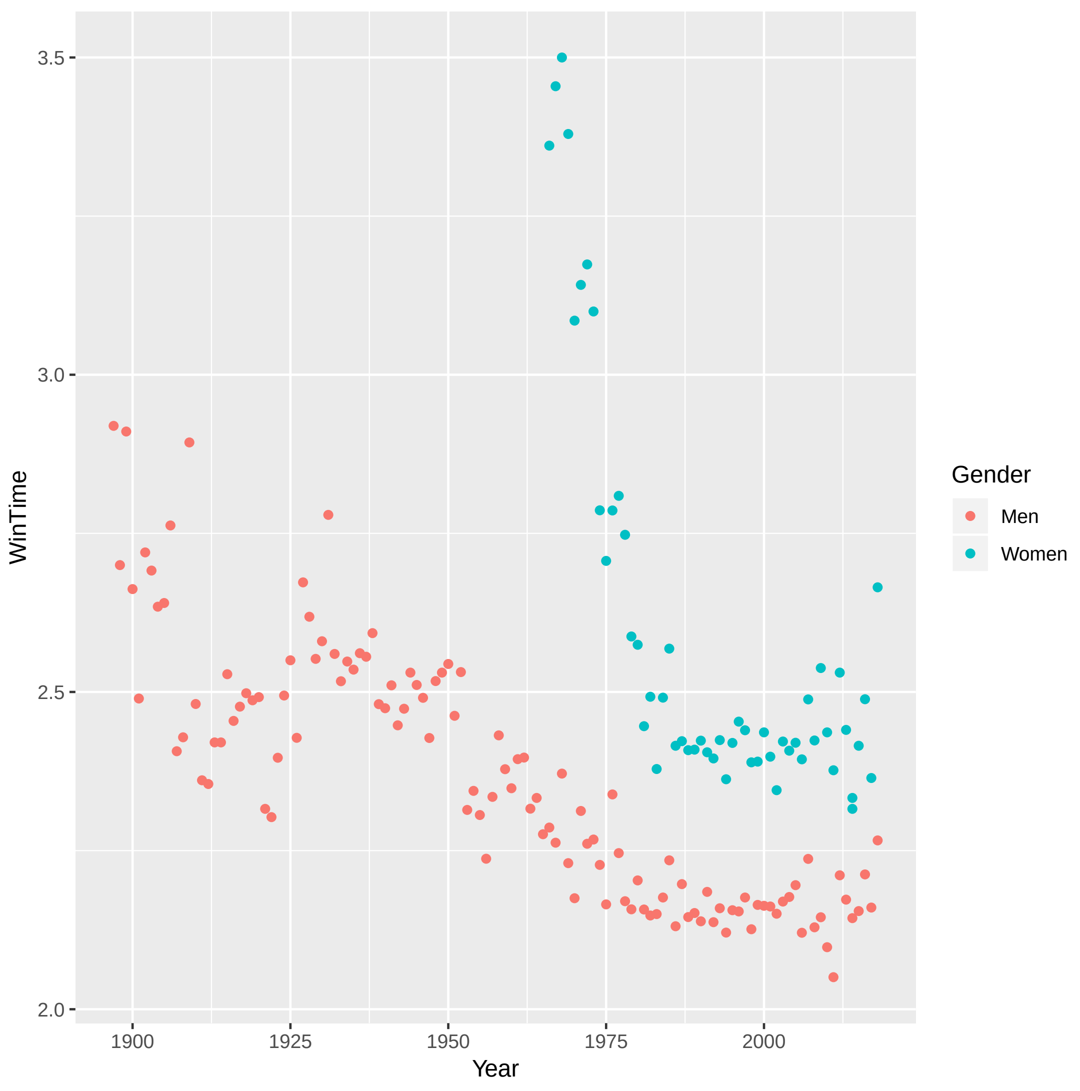
I am writing this on July 25, 2018, so the 2018 races are included (the races take place in April).

Extra: the Wikipedia page `https://en.wikipedia.org/wiki/Marathons_at_the_Olympics` gives a good history of the marathon race in general (which is linked to the modern Olympics). Back in the beginning, it was seen as an unreasonably long distance for a human to run at all, but now there are things like the Ironman, which is a triathlon where the running portion is a marathon (and there are swimming and cycling portions of comparable length before that).

(d) (3 marks) Make a suitable plot of the data.

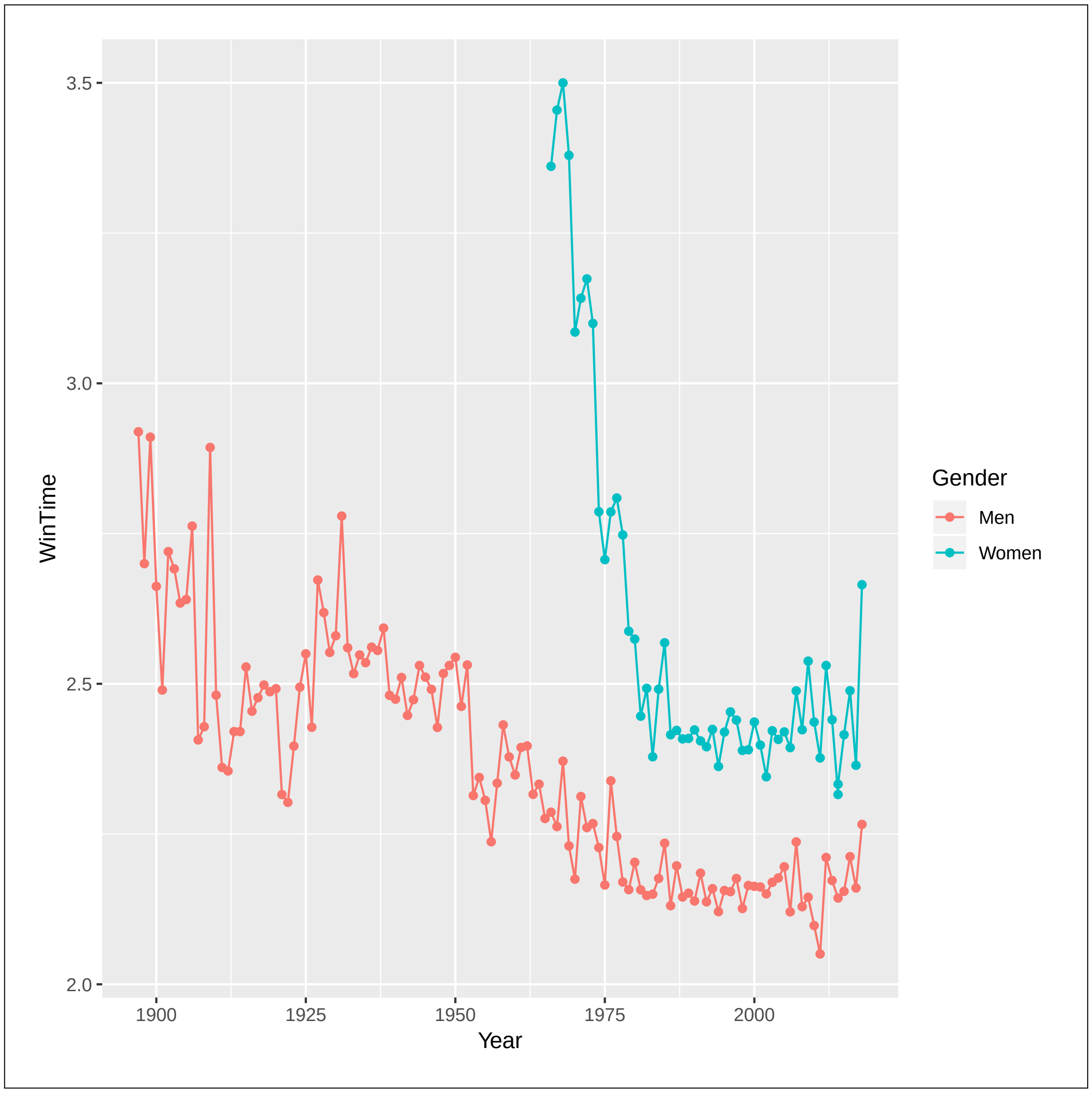
Solution: I think there is really only one way to do this. First, look at your variables: gender (categorical), year (quantitative), winning time (quantitative). Two quantitative and one categorical; this suggests a scatterplot of the two quantitative variables, with the categorical `Gender` distinguished by colour. This is a plot over time, so tradition decrees that the time, `Year`, goes on the x-axis:

```
ggplot(marathon,aes(x=Year, y=WinTime, colour=Gender))+
    geom_point()
```



Since these are time series, you could also reasonably join each point to the next one, which is alarmingly simple:

```
ggplot(marathon,aes(x=Year, y=WinTime, colour=Gender))+
    geom_point()+geom_line()
```



(e) (3 marks) What do you conclude from your plot?

Solution: Think about what we are trying to learn, and what the plot tells you about that. We are looking for relationships between winning time and year, and we should probably address that for each gender separately.

Men's marathon winning times (in red) have shown a pretty steady decline over the years, with the exception of some quicker races between about 1910 and 1925. You might be happy to fit a straight line to these. Or you could say that the winning times have levelled off since the mid-1980s after a steady decline. I'd go for that too.

Women's marathon times declined very fast until about 1980, and then have remained more or less steady since then. This trend does not look linear at all.

When the women started running the Boston Marathon, they were a lot slower than the men, but the gap quickly closed, and now the women's winner is only about 15 minutes slower than the men's winner. (That means that the fastest women have been faster than *a lot* of men in any year since about 1985.)

I think it's reasonable for you to look (at least briefly) at those three things: (i) the men's times, (ii) the women's times, (iii) some kind of comparison between men and women.

I didn't ask you to come up with reasons for these trends, but you can speculate if you like. The usual reasons for improvements in athletic performance over time are things like diet, training and equipment; you can imagine there would be steady improvements in these over time, which would explain the steady improvement in men's performances. The story for women looks different; my take is that back in the 1960s, there were very few women running marathons at all, and in the first few years after Boston had a women's race, the number would have increased sharply, and the whole field of women's marathon running would have become more competitive, with advances in training and all the other things. The Olympics didn't have a women's marathon race until 1984 (I remember when the longest race for women was 3000m), so you could imagine that the introduction of this would have greatly increased the popularity of the marathon among women runners.

Winning times vary from year to year for other reasons. In earlier years, the course was changed from one year to the next (becoming slightly longer or shorter), and in a race as long as the marathon, weather conditions play a role, with it being harder to run fast if it is warm or if it is raining. (Both 2018 winning times were quite a bit slower than the 2017 times, suggesting bad weather in 2018.)

(f) (5 marks) For the women's winning times, fit a straight line predicting the winning time from the year, and decide whether this is an appropriate thing to be doing.

Solution: This is deliberately left open-ended, because part of the work is for you to figure out what needs to be done and how to do it.

I think there are these steps:

- extract just the women's times from the data frame
- fit a regression predicting winning time from year
- assess the residuals from this regression.

You should describe clearly what you are doing, so that the grader can follow your process. Expect to lose marks if your work, even if properly done, is not clearly motivated.

First, we have to extract the women's data only. This will likely involve `filter` in some guise:

```
(marathon %>%
  filter(Gender=="Women") ->
women)
```

```
## # A tibble: 54 x 3
##    Gender  Year WinTime
##    <chr>  <int>   <dbl>
##  1 Women   1966    3.36
##  2 Women   1967    3.45
##  3 Women   1968    3.5
##  4 Women   1969    3.38
##  5 Women   1970    3.09
##  6 Women   1971    3.14
##  7 Women   1972    3.17
##  8 Women   1973    3.10
##  9 Women   1974    2.79
## 10 Women   1975    2.71
## # ... with 44 more rows
```

Or put `women=` at the front. I surrounded the whole thing with brackets to both save and display the result (to check that I have the right thing), but as long as it's obvious at some point that you have the right data, I'm not insisting on this.

Next, for these data, fit a regression predicting winning time from year, and display the results:

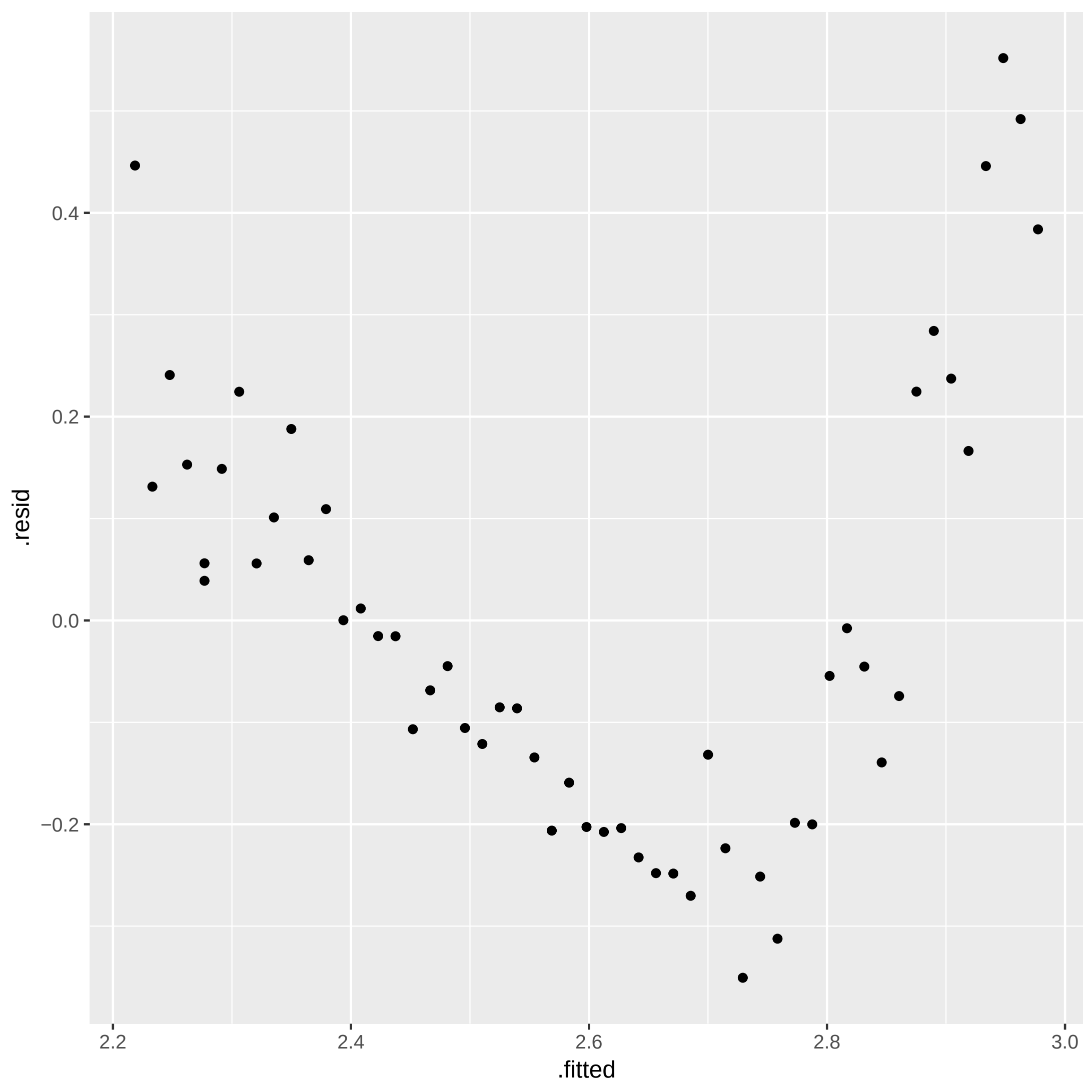
```
women.1=lm(WinTime~Year,data=women)
summary(women.1)
```

```
##
## Call:
## lm(formula = WinTime ~ Year, data = women)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -0.35064 -0.18873 -0.04504  0.14439  0.55188
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept) 31.663478   3.871991   8.178 6.68e-11 ***
## Year        -0.014591   0.001943  -7.508 7.66e-10 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.2205 on 52 degrees of freedom
## Multiple R-squared:  0.5202,Adjusted R-squared:  0.511
## F-statistic: 56.38 on 1 and 52 DF,  p-value: 7.665e-10
```

There is a downward trend with strong significance, but R-squared at 0.52 is not so big. This could be for a couple of reasons: (i) the straight line fits OK, but there is a lot of variability, or (ii) the relationship is *not* a straight line, and some other model could be a lot better. R-squared *does not* distinguish between these, so you *need* to look at a residual plot to figure out what's going on. (You probably have some suspicions from the first plot you drew.)

So, the residual plot (for you, after some discussion about "checking whether the straight-line model is appropriate"):

```
ggplot(women.1,aes(x=.fitted,y=.resid))+geom_point()
```



That is as clear a down-and-up curve as you could wish to see, indicating that the relationship between winning time and year is definitely *not* a straight line.

This is as far as I was intending you to go (it is enough stuff to give you on an assignment), but you probably now have two nagging questions:

1. what kind of relationship *is* this?
2. what about the men?

Extra, therefore, is trying to gain some insight on these.

One of the things limiting athletic performance is the human body, so that even with all the improvements in diet, training, etc., a human will never be able to run a marathon in, say, less than an hour.[29] This idea of things approaching a limit is what an asymptote model is good for, so we can try that here:

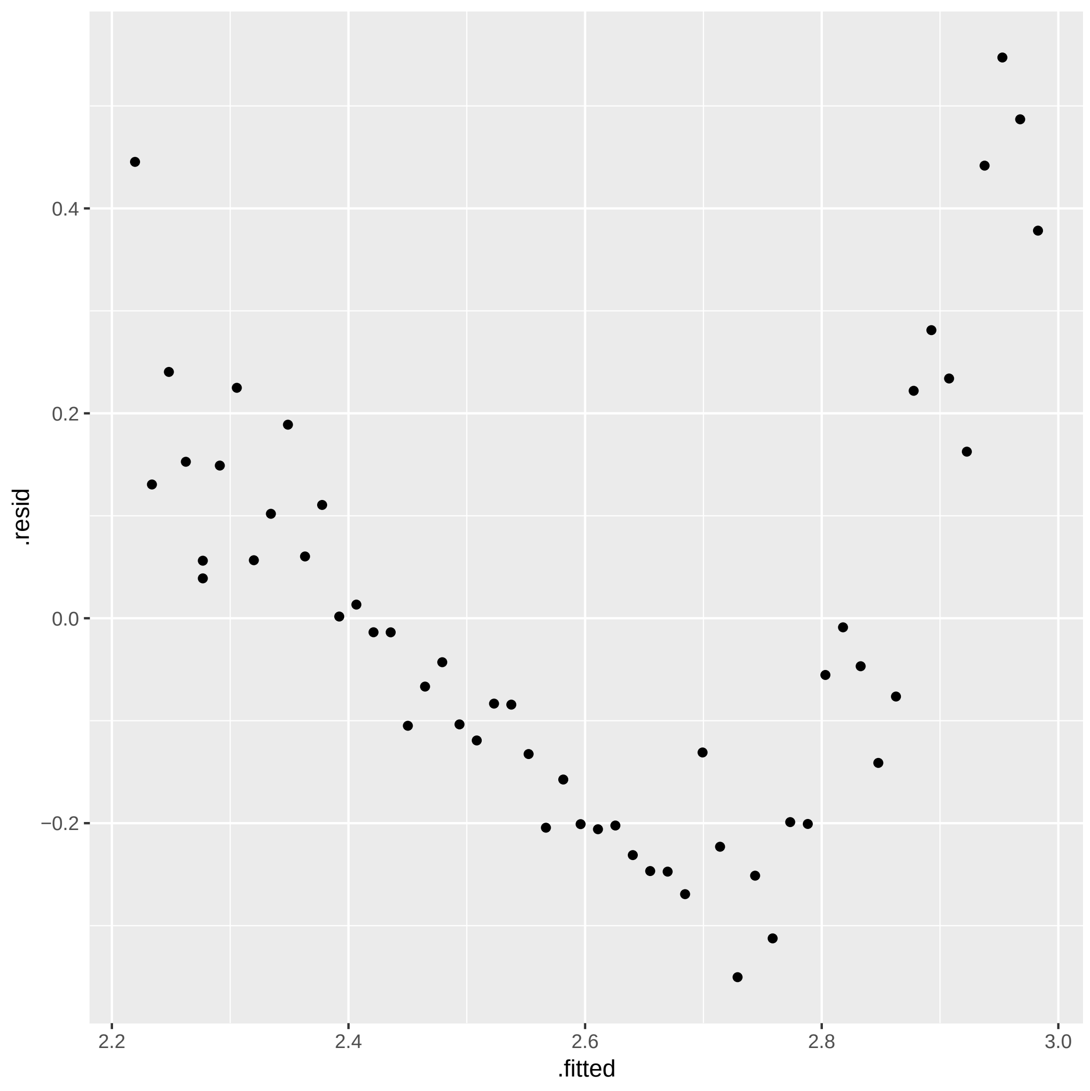
```
women.2=lm(WinTime~I(Year^(-1)),data=women)
summary(women.2)
```

```
##
## Call:
## lm(formula = WinTime ~ I(Year^(-1)), data = women)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -0.35027 -0.18854 -0.04484  0.14442  0.54729
##
## Coefficients:
##               Estimate Std. Error t value Pr(>|t|)
## (Intercept)    -26.637      3.847  -6.924 6.54e-09 ***
## I(Year^(-1)) 58232.336   7664.015   7.598 5.52e-10 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.2192 on 52 degrees of freedom
## Multiple R-squared:  0.5261,Adjusted R-squared:  0.517
## F-statistic: 57.73 on 1 and 52 DF,  p-value: 5.518e-10
```

This is not much of an improvement R-squared-wise. Also, the intercept in this kind of model is the limit (as “year tends to infinity”), and this value doesn’t make much sense (we’d expect it to be some number of hours a bit less than 2, not less than zero like this.)

So the residuals may not be much improved either:

```
ggplot(women.2,aes(x=.fitted,y=.resid))+geom_point()
```



Basically unchanged. So this model is no improvement.

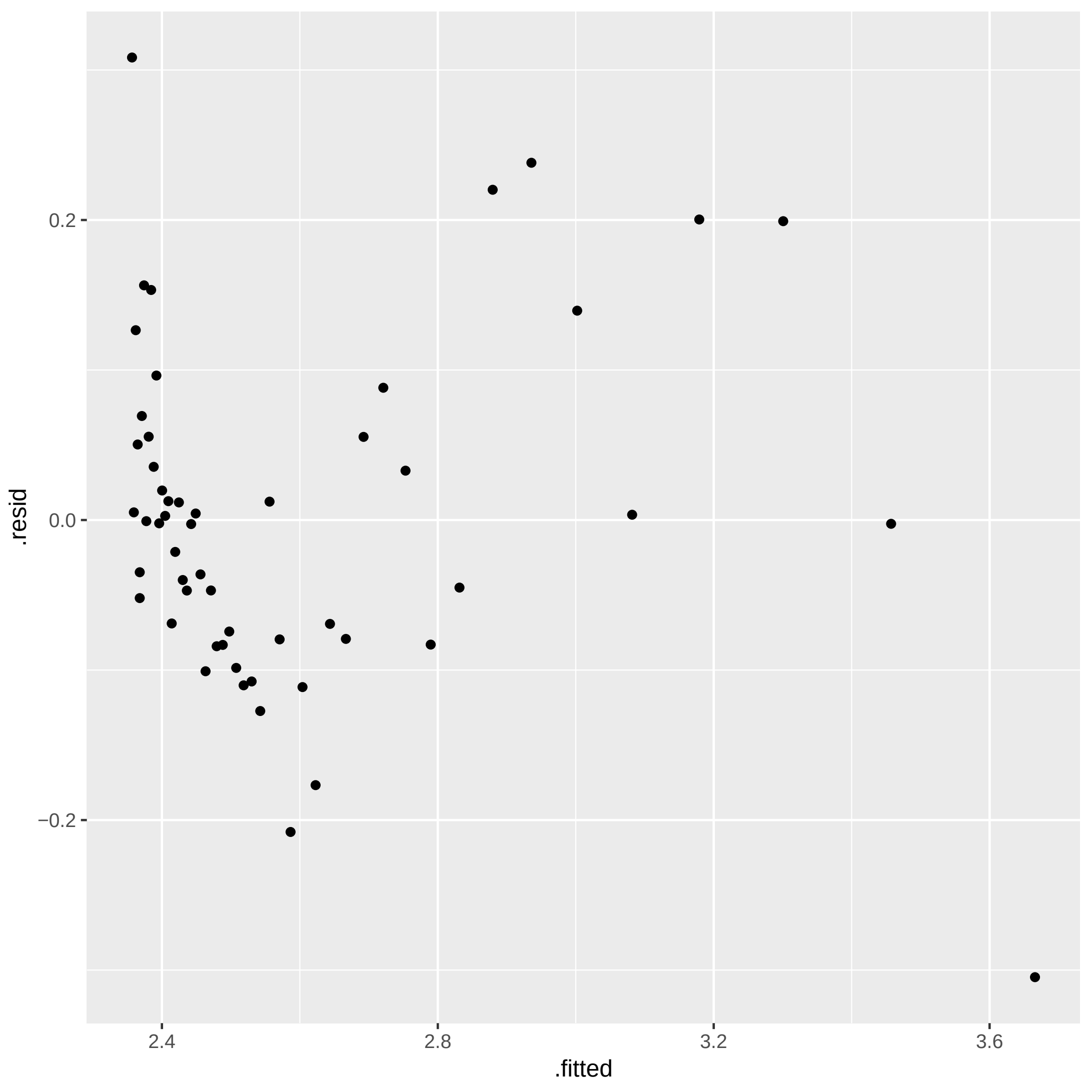
How about setting the zero year to be something closer to the `Year` values in the data, like 1960? For a linear model, it wouldn't make any difference, but this relationship is non-linear, so it might make a difference where on the curve we are:

```
women.3=lm(WinTime~I((Year-1960)^(-1)),data=women)
summary(women.3)
```

```
##
## Call:
## lm(formula = WinTime ~ I((Year - 1960)^(-1)), data = women)
##
## Residuals:
##       Min        1Q    Median        3Q       Max
## -0.304719 -0.077999 -0.002341  0.054169  0.308339
##
## Coefficients:
##                       Estimate Std. Error t value Pr(>|t|)
## (Intercept)            2.20560    0.02623   84.08   <2e-16 ***
## I((Year - 1960)^(-1))  8.76136    0.47519   18.44   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.116 on 52 degrees of freedom
## Multiple R-squared:  0.8673,Adjusted R-squared:  0.8648
## F-statistic: 339.9 on 1 and 52 DF,  p-value: < 2.2e-16
```

Well, that's a big improvement! How do the residuals look?

```
ggplot(women.3,aes(x=.fitted,y=.resid))+geom_point()
```



Better, if perhaps not perfect. The intercept of 2.20 makes good sense as a limit, too (the women's winning times are approaching that without having reached it).

I persisted with this form of model because I thought it had some hope of fitting well and appropriately, if I tweaked it.

All right, now the men. I read the Wikipedia page and learned that after the 1924 race, the starting point was changed and the race was made a little longer. So let's start at 1925:

Start with a straight line:

```
(marathon %>%
    filter(Gender=="Men", Year>=1925) ->
men)
```

```
## # A tibble: 94 x 3
##    Gender  Year WinTime
##    <chr>  <int>   <dbl>
##  1 Men     1925    2.55
##  2 Men     1926    2.43
##  3 Men     1927    2.67
##  4 Men     1928    2.62
##  5 Men     1929    2.55
##  6 Men     1930    2.58
##  7 Men     1931    2.78
##  8 Men     1932    2.56
##  9 Men     1933    2.52
## 10 Men     1934    2.55
## # ... with 84 more rows
```

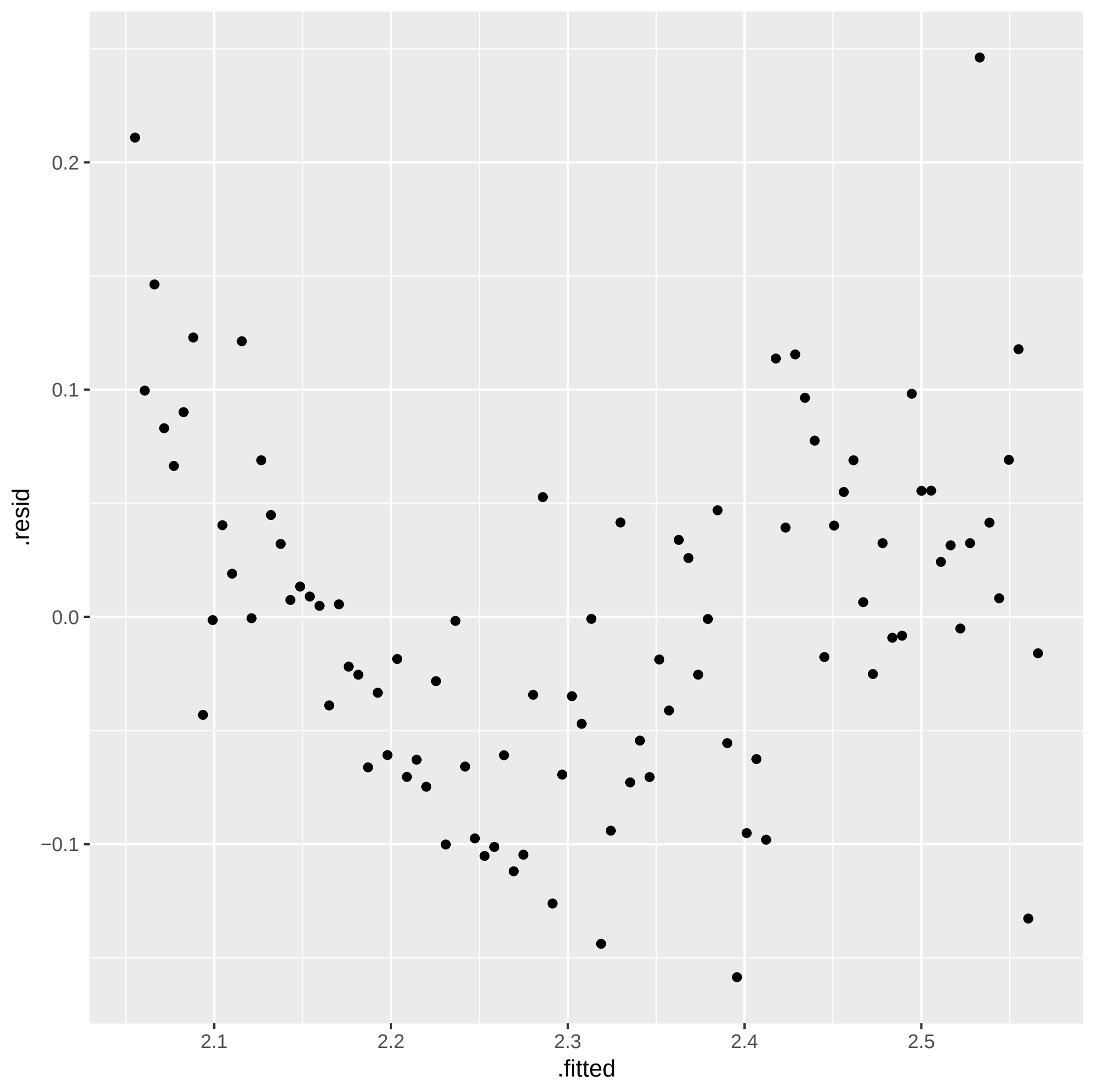
```
men.1=lm(WinTime~Year,data=men)
summary(men.1)
```

```
##
## Call:
## lm(formula = WinTime ~ Year, data = men)
##
## Residuals:
##       Min        1Q    Median        3Q       Max
## -0.158498 -0.059466 -0.001142  0.044024  0.246146
##
## Coefficients:
##               Estimate Std. Error t value Pr(>|t|)
## (Intercept) 13.1380892  0.5736514   22.90   <2e-16 ***
## Year        -0.0054920  0.0002909  -18.88   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.07654 on 92 degrees of freedom
## Multiple R-squared:  0.7948,Adjusted R-squared:  0.7926
## F-statistic: 356.3 on 1 and 92 DF,  p-value: < 2.2e-16
```

The linear model fits well (for this kind of data).

How are the residuals here?

```
ggplot(men.1, aes(x=.fitted, y=.resid))+geom_point()
```



This is a down-and-up curve, not as clear as for the women (which is why I had you investigate the women's times), but a curve nonetheless.

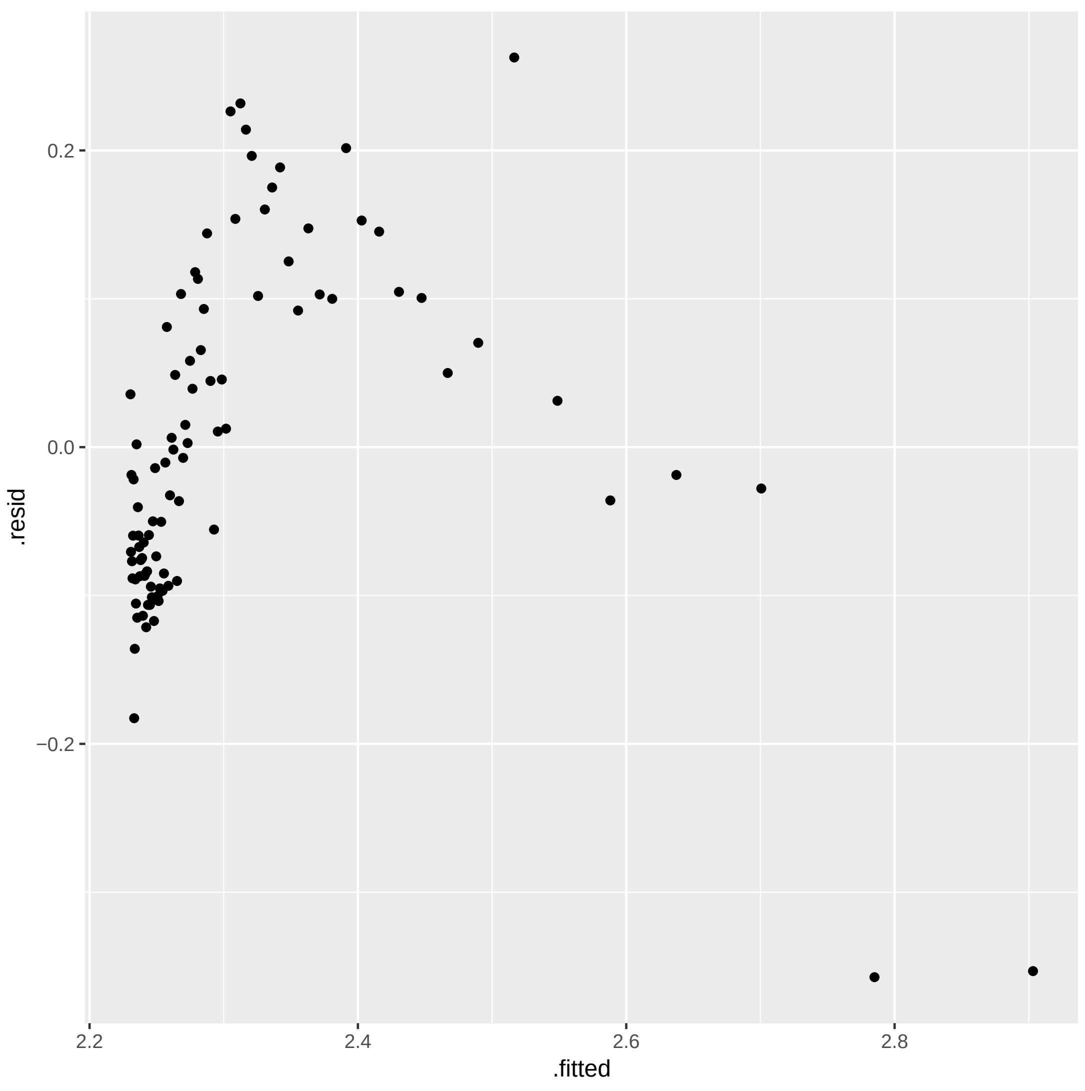
Let's try an asymptote model, subtracting 1920 from all the years:

```
men.2=lm(WinTime~I((Year-1920)^(-1)),data=men)
summary(men.2)
```

```
##
## Call:
## lm(formula = WinTime ~ I((Year - 1920)^(-1)), data = men)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -0.35724 -0.08538 -0.01873  0.09288  0.26265
##
## Coefficients:
##                       Estimate Std. Error t value Pr(>|t|)
## (Intercept)            2.19431    0.01662  132.00   <2e-16 ***
## I((Year - 1920)^(-1))  3.54425    0.35072   10.11   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.1163 on 92 degrees of freedom
## Multiple R-squared:  0.5261,Adjusted R-squared:  0.5209
## F-statistic: 102.1 on 1 and 92 DF,  p-value: < 2.2e-16
```

This fits worse than the linear model. But we might want to go with it anyway, if the residual plot looks better:

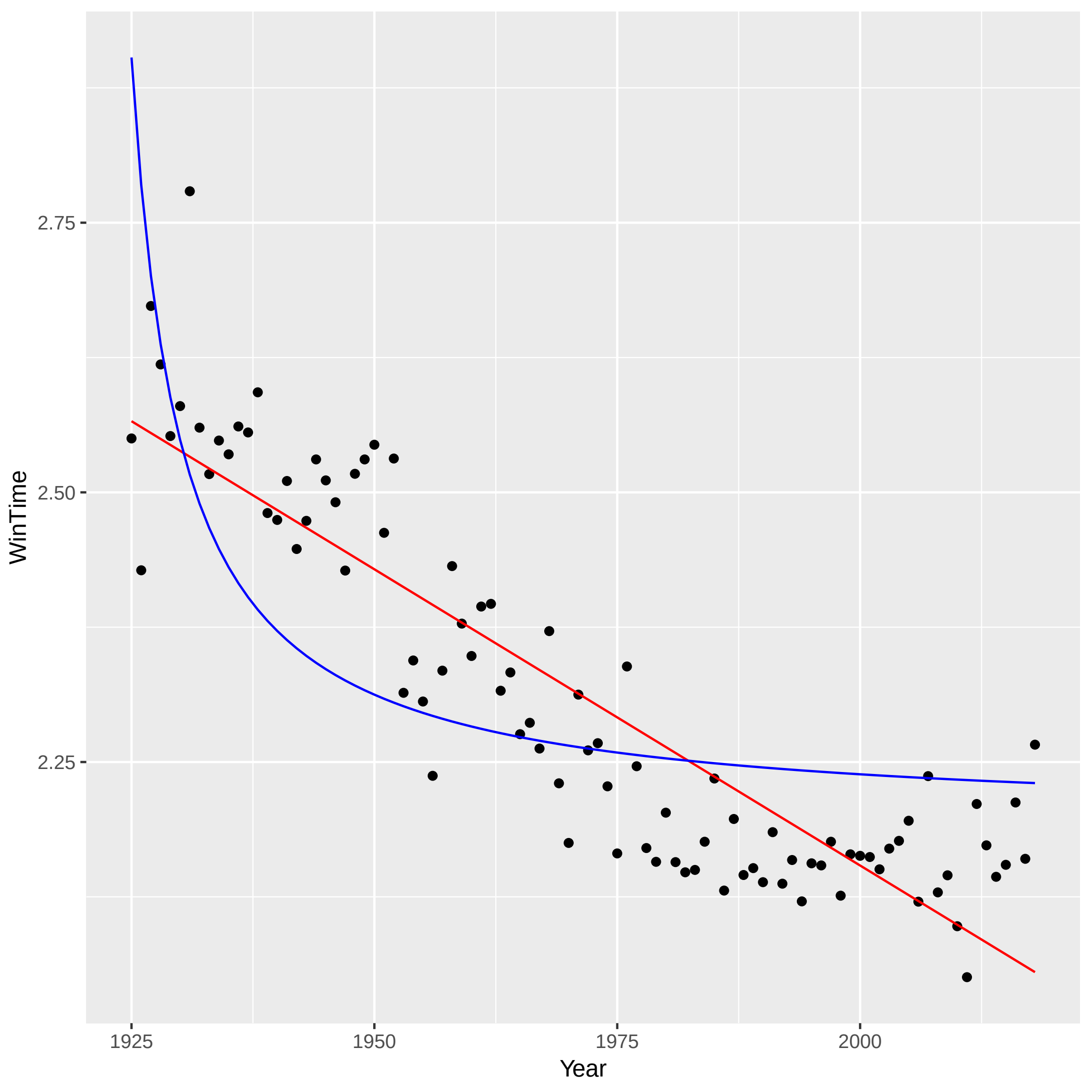
```
ggplot(men.2,aes(x=.fitted,y=.resid))+geom_point()
```



That looks like a sharp upward trend and then a gradual downward one. In this case, I like the asymptote model a lot less than the linear one. (This is why I didn't give you the men's times to work with, since then *you* would have been facing this.) The suggestion is that my asymptote model is the wrong kind of curve.

Maybe I can plot the data with both sets of fitted values, and see how it looks:

```
ggplot(men,aes(x=Year,y=WinTime))+geom_point()+
    geom_line(aes(y=fitted(men.1)),colour="red")+
    geom_line(aes(y=fitted(men.2)),colour="blue")
```



The straight line (red) is not awful (as evidenced by its decent R-squared), but there seems to be some levelling off at the end that the line won't capture (because it keeps going down at the same rate). The curve (blue) does indeed seem to be the wrong shape: it bends too much at the beginning (eg. the prediction for 1950 is way too low), whereas it needs to descend more gradually and then level off.

In this case, we need to think about another aspect of the asymptote model: not just where its limit is, but *where it starts*. In the case of the women, pretty much no-one was running marathons in 1960, so it made sense to subtract 1960 from all the years. For the men here, I arbitarily subtracted 1920 from all the years, but men had been running marathons for some years before that. If you look back to the first graph you drew, the women had a rapid descent after about 1960 (or 1965), so that makes sense as a starting point; the men never really had such a rapid decline, but if one happened, it would have been between about 1900 and 1915, before the course length changed.

`fitted()` with the name of a model inside is another way to get the fitted values from a model. My strategy here is to make a scatterplot of the points as usual, and then on that overlay the fitted line and curve. The fitted line and curve are using the same `Years` but different fitted values each time, so I have to overwrite the `y` for each of them with the right thing, and make them different colours so that I can tell them apart. The fitted line/curve are plotted as "lines" rather than points to show the trends and to distinguish them from the data. (They are actually drawn as a lot of close-together points joined by lines, but there are so many years that you don't see the straight-line pieces they are made out of.)

The asymptote on the asymptote model clearly makes no sense: it should be somewhere down below 2 hours, not 2.22 hours as the model gives us.

I previously tried this with all the men's winning times going back to 1897, and neither of these two models fit very well. (The winning times decreased until 1924, then jumped up at 1925 and decreased after that, which is not what our model is fitting.)

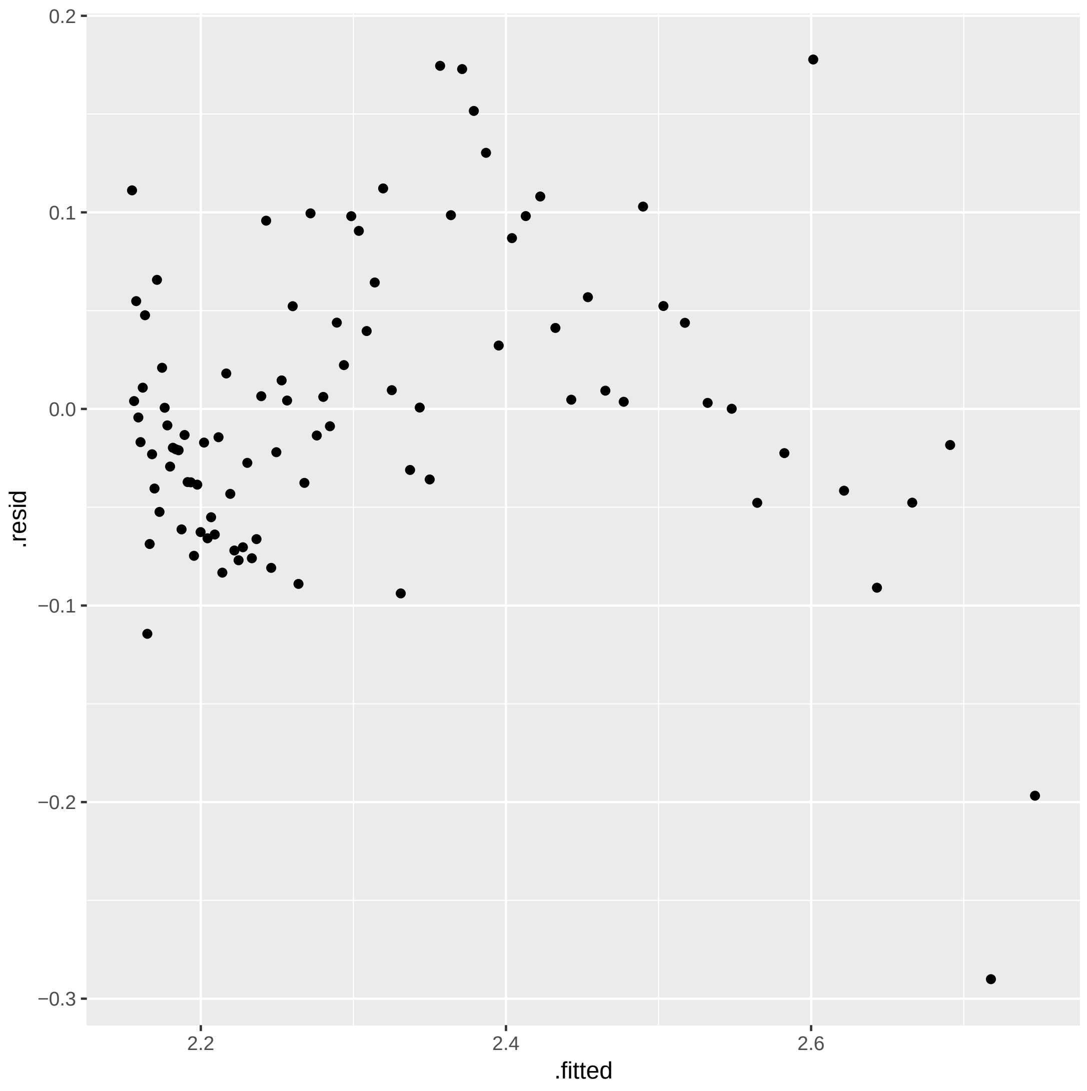
Let me have one more go at the men's times, which is to fit an asymptote model subtracting 1900 from all the years:

```
men.3=lm(WinTime~I((Year-1900)^(-1)),data=men)
summary(men.3)
```

```
##
## Call:
## lm(formula = WinTime ~ I((Year - 1900)^(-1)), data = men)
##
## Residuals:
##       Min        1Q    Median        3Q       Max
## -0.290071 -0.042786 -0.008573  0.043893  0.177775
##
## Coefficients:
##                        Estimate Std. Error t value Pr(>|t|)
## (Intercept)             1.99581    0.01831  109.03   <2e-16 ***
## I((Year - 1900)^(-1)) 18.77293    0.98641   19.03   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.07604 on 92 degrees of freedom
## Multiple R-squared:  0.7974,Adjusted R-squared:  0.7952
## F-statistic: 362.2 on 1 and 92 DF,  p-value: < 2.2e-16
```

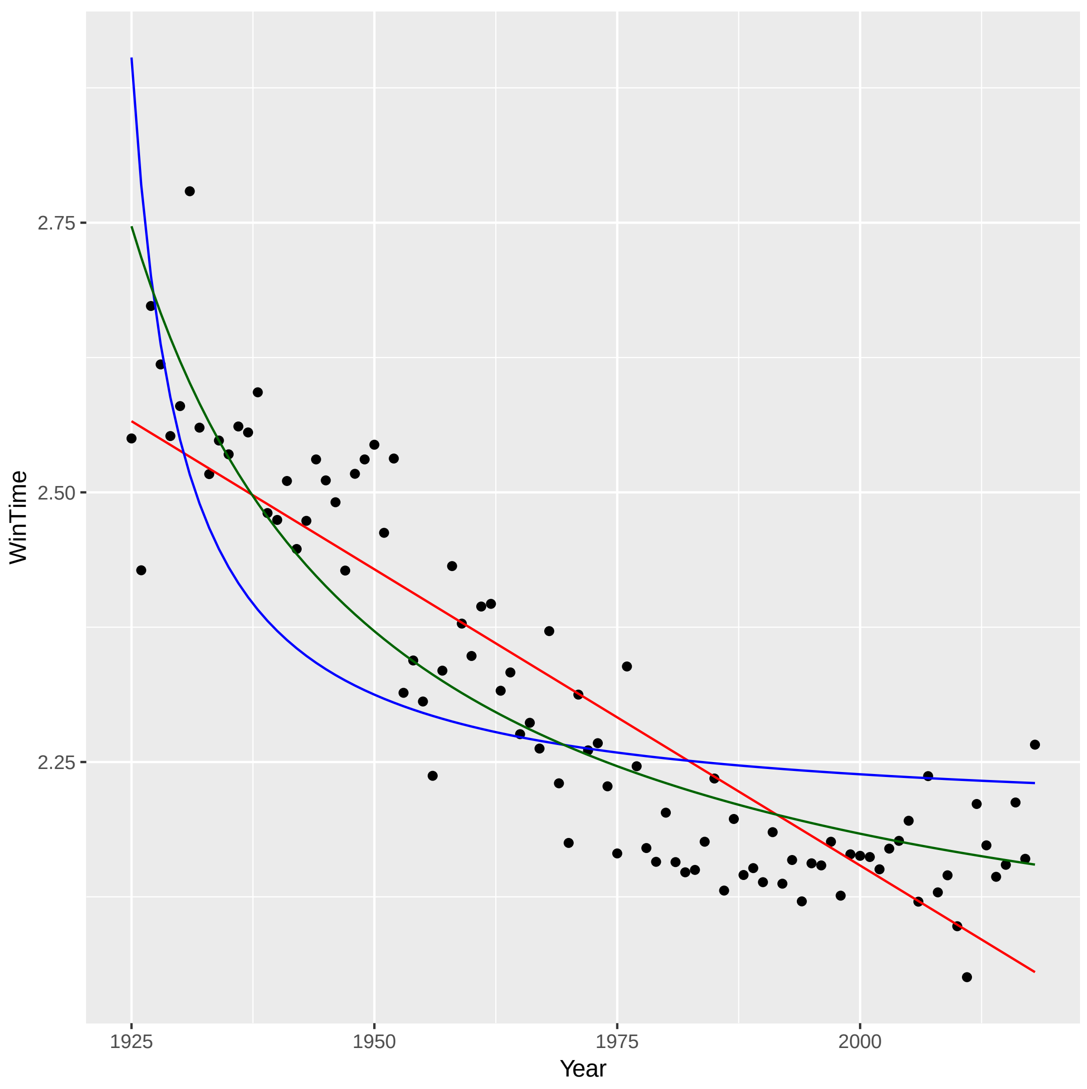
This fits about as well as our linear model did (and better than the other asymptote model), and so if its residual plot looks OK, this would be the best model so far:

```
ggplot(men.3,aes(x=.fitted,y=.resid))+geom_point()
```



That's a bit of a curve, but better than we have seen before. Let's try plotting the data with all *three* fitted curves and see how they stack up:

```
ggplot(men,aes(x=Year,y=WinTime))+geom_point()+
    geom_line(aes(y=fitted(men.1)),colour="red")+
    geom_line(aes(y=fitted(men.2)),colour="blue")+
    geom_line(aes(y=fitted(men.3)),colour="darkgreen")
```



The green curve is the one where we subtracted 1900. While it's surely possible to do better, this is definitely the best we have so far. (It seems to overestimate the winning times in the 1980s, but it gets the ones since 2000 about right.) If you look back at the summary for `men.3`, the intercept is just under 2 hours, which seems about right for the limiting time: no-one has beaten 2 hours yet, the closest being 2:03.[30]

Is it possible to *estimate* the starting point? A little math to think about this: let y be the winning time and x be the year. Then we are fitting a model of this form:

$$y = a + b/(x - c)$$

When we were thinking about things like exponential growth, we could rewrite the model to be linear in a, b, c, with functions of x and y (logs in that case). But we can't do that here, so we are stuck with the "guess-and-check" idea that I used here.

(g) (3 marks) Add a Conclusion to your report that expresses what you have learned so far, and says briefly what you would work on next.

Solution: Talk about how you analyzed the women's times, and that you found a downward but curved trend. There are two obvious directions that you might go, given more time: analysis of the curved trend in the women's times, along the lines of what I did in the Extra, or analysis of the men's times (the most immediate thing being to discover if the trend there is curved as well, as I found it was).

Say *something* reasonably sensible about what you have discovered so far (2 points) and where you might go next (1 point).

In with this, the grader will award you up to three points for your writing for this question (and whether you have tried to make it look like a report).

References

[1] D. J. Best & J. C. W. Rayner (1987) Welch's Approximate Solution for the Behrens–Fisher Problem, Technometrics, 29:2, 205-210, DOI: 10.1080/00401706.1987.10488211

Notes

[1]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 known as "machine epsilon", the smallest number on a computer that is distinguishable from zero.

[2]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.

[3]see discussion elsewhere about Yates' Correction and fixed margins.

[4]This was a year ago when I first wrote this

[5]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.

[6]It's probably better in a report to use language a bit more formal than "a bunch". Something like "a number" would be better.

[7]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.

[8]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.

[9]The test goes back to the 1940s.

[10]I found this by googling, after I had scrolled past all the pages of articles about the baseball pitcher who *lives* in a van.

[11]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.

[12]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.

[13]Sometimes the column playing the role of "rep" *is* interesting to us, but not here.

[14]To allow for the fact that measurements on the same subject are not independent but correlated.

[15]And then thrown away.

[16]It needs `print` around it to display it, as you need `print` to display something within a loop or a function.

[17]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.

[18]The packages before *tidyverse* other than *MASS* are all loaded by the *tidyverse*, which is why there are so many.

[19]All the ones that are part of this project, anyway.

[20]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.

[21]Mathematically, $e^x \simeq 1 + x$ 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.

[22]Which was the title of a song by Prince.

[23]As, for example, when Prince died.

[24]When this was graded, it was 3 marks, to clue you in that there are three things to say.

[25]The `summary` output is more designed for looking at than for extracting things from.

[26]These days, there are apps that will let you do this with your phone. I found one called Clinometer. See also `https://gabrielhemery.com/how-to-calculate-tree-height-using-a-smartphone/`.

[27]Ahem.

[28]This is also what SAS does.

[29]Mind you, some people said the same thing about running the mile in less than 4 minutes, and then Roger Bannister did it in 1954. His record was broken shortly afterwards, so it wasn't any kind of limit after all.

[30]This time is the fastest ever run for a marathon, but is *not* the world record. Races up to 10,000m are on running tracks, which are more or less comparable one with another, barring things like wind (and there are rules about that), and so the fastest time *is* the world record, but marathons are run on streets rather than around and around and around a 400m oval, so every marathon course is different. The international federation has rules about when they will accept a marathon world record, which the Boston marathon doesn't meet. One of the problems about the Boston course is that if the wind is blowing the right way, the runners will have the wind at their backs for most of the race, which makes it easier to run faster.