University of Toronto Scarborough
Department of Computer and Mathematical Sciences
STAC32 (K. Butler), Final Exam
December 11, 2024

Aids allowed (on paper, no computers):

- My lecture overheads (slides)
- Any notes that you have taken in this course
- Your marked assignments
- My assignment solutions
- Non-programmable, non-communicating calculator

This exam has 12 numbered pages.

In addition, you have an additional booklet of Figures to refer to during the exam.

The maximum marks available for each question are shown next to the question.

If you need more space, use the last page of the exam. Anything written on the back of the page will not be graded.

You may assume throughout this exam that the code shown in Figure 1 of the booklet of Figures has already been run.

The University of Toronto's Code of Behaviour on Academic Matters applies to all University of Toronto Scarborough students. The Code prohibits all forms of academic dishonesty including, but not limited to, cheating, plagiarism, and the use of unauthorized aids. Students violating the Code may be subject to penalties up to and including suspension or expulsion from the University.

Hawks

A “blind” is a place where people can watch wild animals or birds without being noticed by them. A college maintains a hawk blind at Lake MacBride, Iowa, where students can observe hawk and other birds. Birds of two species of hawk, Cooper’s Hawk (`CH` in the data file) and Red-tailed Hawk (`RT` in the data file) were captured, weighed and then released; the weight was measured in grams. Some of the data file is shown in Figure 2 (the remaining lines are laid out in the same way). The data file is called `hawks.txt`, and is in the same folder in which you are currently running R Studio.

(1) (3 points) What code would read the data from the file into a dataframe called `hawks`, and display at least some of the dataframe that was read in?

The data file is laid out as (depending how you look at it) aligned columns, or as columns with varying numbers of spaces between them. Either way, the right thing is `read_table`:

```r
hawks <- read_table("hawks.txt")
```

```
-- Column specification --------------------------------------------------------
cols(
  species = col_character(),
  weight = col_double()
)
```

```r
hawks
```

```
# A tibble: 642 x 2
   species weight
   <chr>    <dbl>
 1 RT         920
 2 RT         930
 3 RT         990
 4 CH         470
 5 RT        1090
 6 RT         960
 7 RT         855
 8 RT        1210
 9 RT        1120
10 RT        1010
# i 632 more rows
```

Two points for getting the `read_table` right, and one for something that will display at least some of the dataframe (`View(hawks)`, for example, would also work).

Some things that will not work, and earn no points:

- `read_delim` (the data values are not separated by a *single* anything; there are sometimes three and sometimes two spaces between the data values)
- `read_tsv` (if the data values were separated by tabs, the weight values would be aligned on the *left*)
- `read.table` (not taught in this course. Make sure your underscore looks like an underscore and not a dot.)

One of these, followed by something that would correctly display the dataframe, if the reading-in had been correct, is one point total.

(2) (3 points) What code will calculate and display the mean and standard deviation of weight, and the number of hawks observed, for each species of hawk?

As with anything else in this exam, if I ask for code (only), I do not need to see the output (and you will probably not know what the output is). If I want to know what output some code will produce, I will ask you for that.

This is a standard group-by and summarize. The summaries can be in any order, but the `group_by` must be first:

```
hawks %>% group_by(species) %>%
  summarize(n = n(), mean_weight = mean(weight), sd_weight = sd(weight))
```

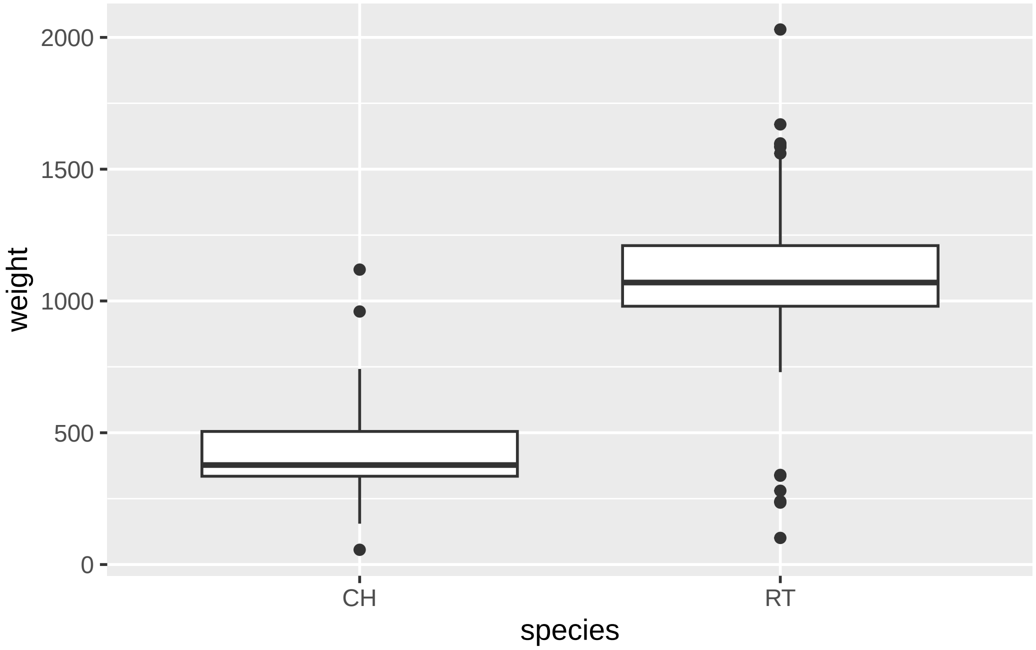
```
# A tibble: 2 x 4
  species     n mean_weight sd_weight
  <chr>   <int>       <dbl>     <dbl>
1 CH         70        420.      162.
2 RT        572       1094.      189.
```

One point for having `group_by` first, and a further two points for getting the `summarize` right. Minus a half point per error or missing item down to a minimum of a half point if something in the `summarize` is correct (for example, having a `summarize` but having nothing else correct).

(3) (2 points) What code will draw a suitable graph of the two variables in this dataframe?

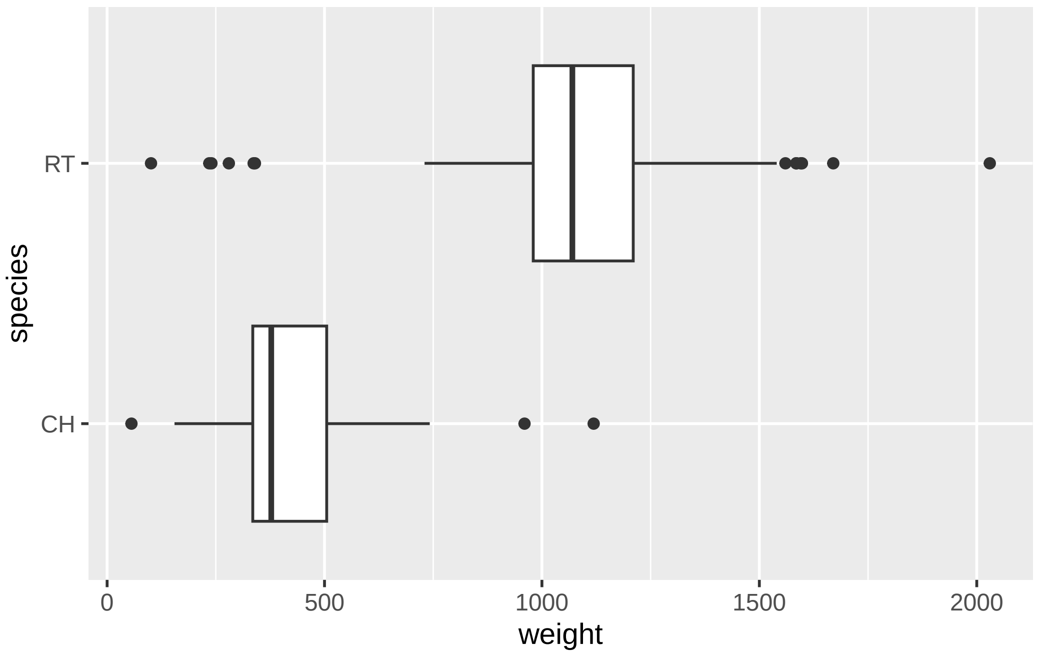
One categorical and one quantitative, so a boxplot (no surprises). Only two points for this rather simple code:

```
ggplot(hawks, aes(x = species, y = weight)) + geom_boxplot()
```



As with the corresponding question on the midterm, having the `x` and `y` the other way around is *wrong* unless you say something that shows you know what you are doing, such as stating that your boxplot will be horizontal rather than vertical and then drawing this:

```
ggplot(hawks, aes(y = species, x = weight)) + geom_boxplot()
```



Seeing this now, you will realize that there are a lot of outliers, and thus if you had seen this, you might have preferred not to run a *t*-test later. But on the exam, you had no way to know.

One out of two for having the `x` and `y` the wrong way around without explanation, minus a half per small error otherwise. Since we are not (at this stage) interested in whether the distributions of weights are *normal*, drawing a (facetted) normal quantile plot is not the best, and earns 1/2 if correctly drawn.

(4) (2 points) What code will display the species and weight of all the hawks that weigh less than 300 grams?

This is choosing some of the hawks (observations), so it needs `filter`. There is no choice of columns (we are displaying all of them), so there is no `select` needed:

```
hawks %>% filter(weight < 300)
```

```
# A tibble: 11 x 2
   species weight
   <chr>    <dbl>
 1 RT         101
 2 CH         183
 3 CH          163
 4 CH          187
 5 RT          280
 6 CH           56
 7 CH          229
 8 CH          155
 9 RT          235
10 RT          240
11 CH          295
```

The boxplot (now that we can see it) indicates that the red-tailed hawks are on average heavier, but there are quite a few red-tailed hawks in among the Cooper's hawks, namely the low-end outliers of the red-tailed hawks.

One point for getting to a `filter` and messing up the condition; minus a half for including a `select` unnecessarily.

(5) (3 points) What code will display only the weights of only the five heaviest Cooper's Hawks?

I think the best way is to grab the Cooper's Hawks first, then find the five heaviest of them:

```
hawks %>%
  filter(species == "CH") %>%
  slice_max(weight, n = 5) %>%
  select(weight)
```

```
# A tibble: 5 x 1
  weight
   <dbl>
1   1119
2    960
3    742
4    640
5    590
```

or sort and then slice:

```
hawks %>%
  filter(species == "CH") %>%
  arrange(desc(weight)) %>%
  slice(1:5) %>%
  select(weight)
```

```
# A tibble: 5 x 1
  weight
   <dbl>
1   1119
2    960
3    742
4    640
5    590
```

Another option is to use `group_by`, but this doesn't quite work:

```
hawks %>%
  group_by(species) %>%
  slice_max(weight, n = 5) %>%
  filter(species == "CH") %>%
  select(weight)
```

```
Adding missing grouping variables: `species`
```

```
# A tibble: 5 x 2
  species weight
  <chr>    <dbl>
1 CH        1119
2 CH         960
3 CH         742
4 CH         640
5 CH         590
```

The grouping variable `species` "comes along for the ride" this way, which we don't want. The correct approach is actually this:

```
hawks %>%
  group_by(species) %>%
  slice_max(weight, n = 5) %>%
  filter(species == "CH") %>%
  ungroup() %>%
  select(weight)
```

```
# A tibble: 5 x 1
  weight
   <dbl>
1   1119
2    960
3    742
4    640
5    590
```

We have seen `ungroup` in the context of `nest_by`, I think. I wouldn't expect you to know that this will be required, so 2.5 if you can get as far as the way without the `ungroup` (since it will not quite work). The filter-first approach is much the safer way to go, because then you are not dealing with the intricacies of grouping, and it will be much clearer to you that it will work.

The first three lines find the five heaviest hawks for *each* species, because when you have groups, things are done within each group:

```
hawks %>%
  group_by(species) %>%
  slice_max(weight, n = 5)
```

```
# A tibble: 11 x 2
   species weight
   <chr>    <dbl>
 1 CH        1119
 2 CH         960
 3 CH         742
 4 CH         640
 5 CH         590
 6 RT        2030
 7 RT        1670
 8 RT        1598
 9 RT        1595
10 RT        1585
11 RT        1585
```

There are eleven lines because this finds the largest five weights in each group *including ties*, and there are two red-tailed hawks that weighed 1585 grams.

(6) (2 points) A dataframe `hawk_names` is shown in Figure 3. What code would use this dataframe, as well as dataframe `hawks`, to create a dataframe that contains the full names of the species of each hawk, along with each hawk's weight? The dataframe produced by your code may contain other columns, and you do not need to save it.

This is looking up the values in one dataframe in another, so it is a join. The simplest is this way:

```r
hawks %>% left_join(hawk_names)
```

```
Joining with `by = join_by(species)`
```

```
# A tibble: 642 x 3
   species weight species_name
   <chr>    <dbl> <chr>       
 1 RT         920 Red-tailed  
 2 RT         930 Red-tailed  
 3 RT         990 Red-tailed  
 4 CH         470 Cooper      
 5 RT        1090 Red-tailed  
 6 RT         960 Red-tailed  
 7 RT         855 Red-tailed  
 8 RT        1210 Red-tailed  
 9 RT        1120 Red-tailed  
10 RT        1010 Red-tailed  
# i 632 more rows
```

It is, as stated in the question, no problem to keep the column `species` as well. (When you are doing one of these outside of an exam, it's a good idea to make sure that the things in the new column(s) you have added do indeed match up with the original columns.)

That said, I think it is actually clearer to explicitly say what you are joining by, ie. this:

```
hawks %>% left_join(hawk_names, join_by(species))
```

```
# A tibble: 642 x 3
   species weight species_name
   <chr>    <dbl> <chr>
 1 RT         920 Red-tailed
 2 RT         930 Red-tailed
 3 RT         990 Red-tailed
 4 CH         470 Cooper
 5 RT        1090 Red-tailed
 6 RT         960 Red-tailed
 7 RT         855 Red-tailed
 8 RT        1210 Red-tailed
 9 RT        1120 Red-tailed
10 RT        1010 Red-tailed
# i 632 more rows
```

The first way works because both dataframes have a column called `species` that is the only column with the same name in both dataframes, so that will be used to join-by. Two points for either of those.

This also works, though, to my mind, more confusingly:

```
hawk_names %>% right_join(hawks)
```

```
Joining with `by = join_by(species)`
```

```
# A tibble: 642 x 3
   species species_name weight
   <chr>   <chr>         <dbl>
 1 CH      Cooper          470
 2 CH      Cooper          324
 3 CH      Cooper          340
 4 CH      Cooper          340
 5 CH      Cooper          475
 6 CH      Cooper          340
 7 CH      Cooper          420
 8 CH      Cooper          365
 9 CH      Cooper          183
```

```
10 CH      Cooper          390
# i 632 more rows
```

The columns come out in a different order, but that is no problem.

I find this more confusing because my mental model is that you have a dataframe with actual data in it, and *then* you have a lookup table (that you got from somewhere else) with the additional information in it. To me, therefore, the data are first, and the lookup table is second.

If you can get the right-join correct, two points for that also.

If you get the join the wrong way around, you get this kind of thing:

```
hawk_names %>% left_join(hawks)
```

```
Joining with `by = join_by(species)`
```

```
# A tibble: 643 x 3
   species species_name  weight
   <chr>   <chr>          <dbl>
 1 SS      Sharp-shinned     NA
 2 CH      Cooper           470
 3 CH      Cooper           324
 4 CH      Cooper           340
 5 CH      Cooper           340
 6 CH      Cooper           475
 7 CH      Cooper           340
 8 CH      Cooper           420
 9 CH      Cooper           365
10 CH      Cooper           183
# i 633 more rows
```

which has an extra row at the top, because `left_join` has at least one row in the result for *everything* in the first dataframe, including the species `SS` that we didn't observe any of (thus the answer has one more row than it should). Doing it the right way around gets us (exactly) one row for each observation in `hawks`, which is exactly what we want. One point only for this, for recognizing that you needed a `join` but not which way around it needed to be. This, therefore, is also one point:

```
hawks %>% right_join(hawk_names)
```

```
Joining with `by = join_by(species)`
```

```
# A tibble: 643 x 3
   species weight species_name
   <chr>    <dbl> <chr>
 1 RT         920 Red-tailed
 2 RT         930 Red-tailed
 3 RT         990 Red-tailed
 4 CH         470 Cooper
 5 RT        1090 Red-tailed
 6 RT         960 Red-tailed
 7 RT         855 Red-tailed
 8 RT        1210 Red-tailed
 9 RT        1120 Red-tailed
10 RT        1010 Red-tailed
# i 633 more rows
```

(7) (1 point) In the dataframe `hawk_names`, suppose the column called `species` had instead been called `abb` (short for "abbreviation"). How would your code from the previous question need to change in order to achieve the same result?

Let's see. First change the column name as given in the question:

```
hawk_names %>% rename("abb" = "species") -> hawk_names
hawk_names
```

```
# A tibble: 3 x 2
  abb   species_name
  <chr> <chr>
1 SS    Sharp-shinned
2 CH    Cooper
3 RT    Red-tailed
```

and then do this:

```
hawks %>% left_join(hawk_names, join_by("species" == "abb"))
```

```
# A tibble: 642 x 3
   species weight species_name
   <chr>    <dbl> <chr>
 1 RT         920 Red-tailed
 2 RT         930 Red-tailed
 3 RT         990 Red-tailed
 4 CH         470 Cooper
 5 RT        1090 Red-tailed
 6 RT         960 Red-tailed
 7 RT         855 Red-tailed
 8 RT        1210 Red-tailed
 9 RT        1120 Red-tailed
10 RT        1010 Red-tailed
# i 632 more rows
```

So your answer needs to be something like: in the `left_join`, include `join_by("species" == "abb")`. This is only one point, so the important thing your answer needs to contain is `join_by("species" == "abb")`.

What ends up happening is that the column name in the result is taken from the first dataframe `hawks` (that is, `species`): the corresponding column in `hawk_names` is temporarily renamed back to `species` and then the lookup is done, so that `abb` seems to disappear (but it actually doesn't).

If you insist on doing this one as a right-join, you have to remember to switch the `join_by` around as well, because "the first dataframe" and "the second dataframe" have switched around:

```
hawk_names %>% right_join(hawks, join_by("abb" == "species"))
```

```
# A tibble: 642 x 3
   abb   species_name weight
   <chr> <chr>         <dbl>
 1 CH    Cooper          470
 2 CH    Cooper          324
 3 CH    Cooper          340
 4 CH    Cooper          340
 5 CH    Cooper          475
 6 CH    Cooper          340
 7 CH    Cooper          420
 8 CH    Cooper          365
 9 CH    Cooper          183
10 CH    Cooper          390
# i 632 more rows
```

The hawks come out in a different order. Because you started off with `hawk_names`, which has Cooper's hawks first, these are the ones that are first in the result. But the same hawks are all there somewhere; for example, the first Cooper's hawk in the previous outputs weighed 470 grams, which is the same hawk as the first one here.

If your answer is `join_by("abb" == "species")`, the grader will need to check back to the previous part and make sure you actually did do a right-join. If you didn't, you only get 0.5.

(8) (3 points) Some code and output is shown in Figure 4. What do you conclude from it, in the context of the data? Explain, showing your logic clearly.

This is a hypothesis test, specifically a two-sample test comparing the weights of the two species of hawks. So:

- null hypothesis: the two species of hawks have the same mean weight
- alternative hypothesis: the two species of hawks have different mean weight (two clues: the output says "not equal", with the bit cut off being "to zero", and there is no `alternative` in the code, so the test must be two-sided)
- The P-value is less that 2.2×10^{-16} (I would accept words like "extremely small")
- The P-value is smaller than 0.05 (or other α of your choosing)
- reject the null in favour of the alternative
- conclude that the mean weights of Cooper's hawks and red-tailed hawks are different.

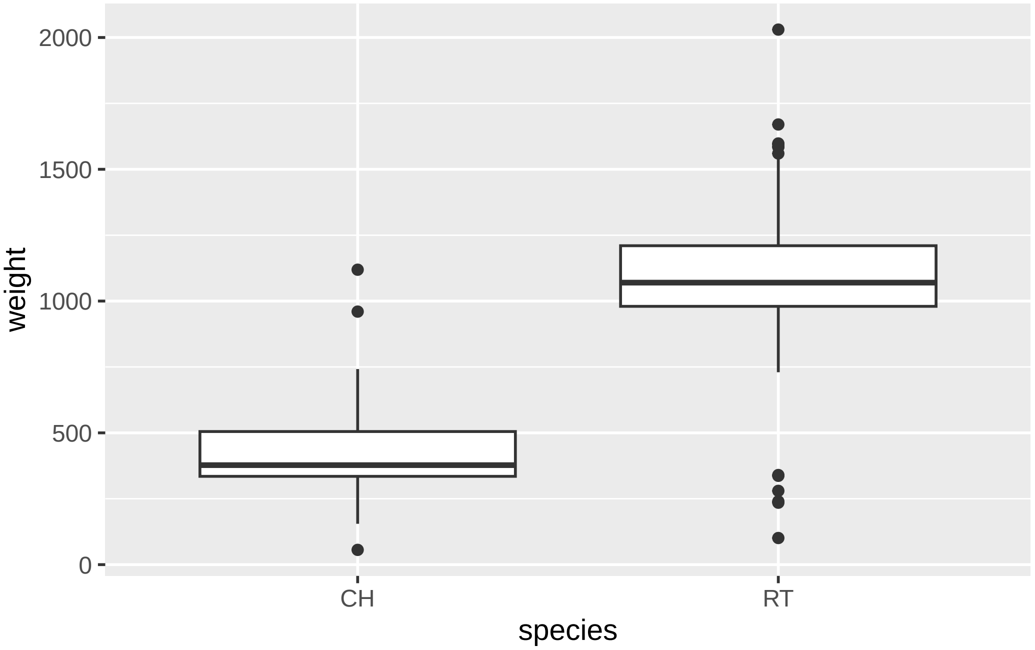
These ingredients need to be in your answer (or strongly implied by it), and it is clearest to follow the steps in something like the way I did: make it clear that you know what the hypotheses are that you are testing, get the P-value, compare it with 0.05, reject the null in favour of the alternative, say what that means so that someone can make a decision on the basis of your conclusion. You need to be clear what is hypothesis and what is conclusion. Writing the hypotheses down *first* and making a conclusion *at the end* is a clear way to keep this straight. This is why I said "showing your logic clearly" in the question; this is not a first-year course in which you write some stuff down and hope that the right answer is in there somewhere, but an upper-level course in which you need to be logically convincing (which is an academic way of saying "show that you know what you are doing").

Extra 1: this is perhaps not a very interesting test to be doing in this context, but the next question for an ornithologist is "*why* are the mean weights different?" This might be a confirmation of something they already know, such as that red-tailed hawks are in general bigger, and thus it is no surprise that they are heavier as part of that (in which case they might have done a one-sided test).

The dataset from which I got these values had all sorts of other information about the hawks they caught, such as whether each one was adult or young and male or female, along with a lot of other body measurements. You might imagine that you could use this information to see what distinguishes the species (there were three of them in the original dataset, the third one being "Sharp-shinned" that appeared in `hawk_names`). Ornithologists are usually pretty good at distinguishing bird species by seeing the birds in flight, but if the species are hard to distinguish, it might be possible to catch a hawk, make the measurements, decide what species it is on the basis of the measurements, and then let it go again. This is the domain of discriminant analysis, which we see in D29. I rather like this dataset for that purpose, so you might be seeing the complete dataset, from which the data in Figure 2 were taken, again later.

Extra 2: let's go back to the boxplots:

```
ggplot(hawks, aes(x = species, y = weight)) + geom_boxplot()
```



There seem to be a lot of outliers here, and most of them seem to be real outliers in that they are clearly out beyond the ends of the whiskers (except for the ones near the end of the upper whisker in the Red-tailed Hawks). Our samples (now that we have seen the output from our group-by and summarize) are on the big side:

```
hawks %>% count(species)
```

```
# A tibble: 2 x 2
  species     n
  <chr>   <int>
1 CH         70
2 RT        572
```

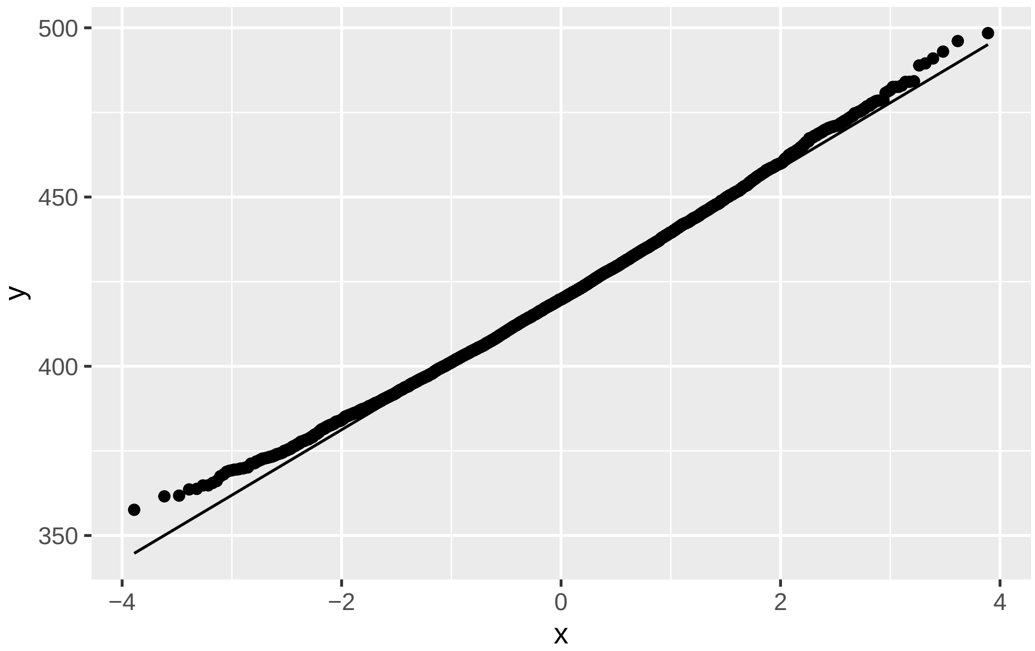
The Cooper's Hawks distribution of weights seems less bad, but it also goes with the smaller sample size, so it is not clear whether these distributions are really normal enough given the sample sizes. But we know how to address that: bootstrap sampling distributions of the sample means.

Let's do the Cooper's Hawks first:

```
hawks %>% filter(species == "CH") -> coopers
```

and then the usual:

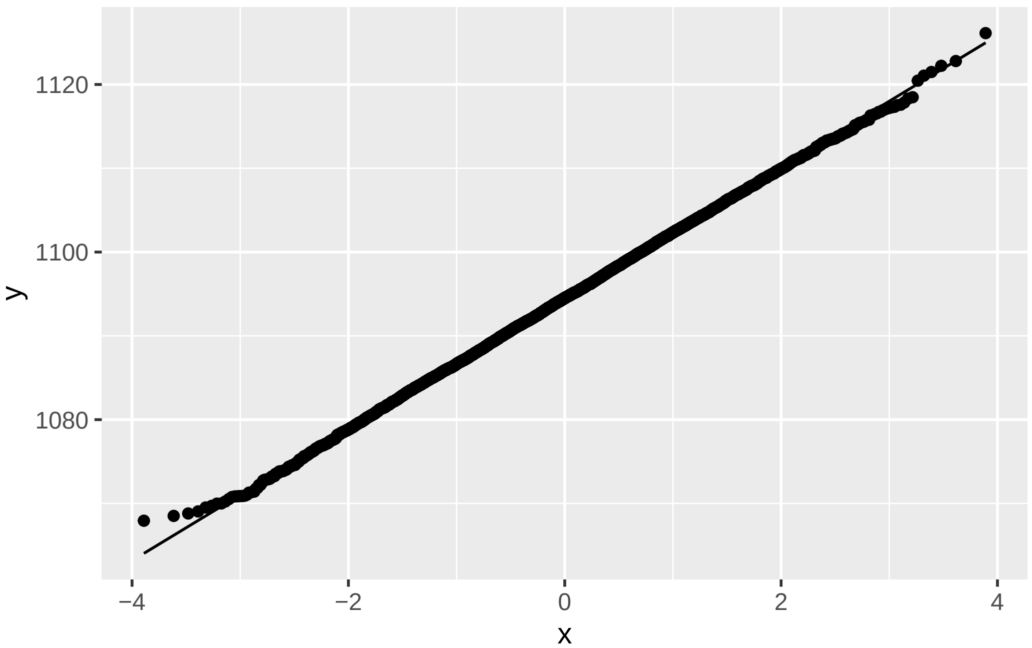
```
tibble(sim = 1:10000) %>%
  rowwise() %>%
  mutate(my_sample = list(sample(coopers$weight, replace = TRUE))) %>%
  mutate(my_mean = mean(my_sample)) %>%
  ggplot(aes(sample = my_mean)) + stat_qq() + stat_qq_line()
```



The original distribution of weights for these had three not-severe outliers, two at the top and one at the bottom, and this has created a sampling distribution that is *very* slightly skewed to the right, but, in the grand scheme of things, very close to normal nonetheless.

The red-tailed hawks go the same way, with much copying and pasting:

```r
hawks %>% filter(species == "RT") -> reds
tibble(sim = 1:10000) %>%
  rowwise() %>%
  mutate(my_sample = list(sample(reds$weight, replace = TRUE))) %>%
  mutate(my_mean = mean(my_sample)) %>%
  ggplot(aes(sample = my_mean)) + stat_qq() + stat_qq_line()
```



This distribution had at least five severe outliers (four at the bottom and at least one at the top), but it also had a very large sample size, over 500 observations. The bootstrap sampling distribution of the sample mean is, in fact, very much normal even though the original distribution was far from it, and that shows that the sample size of over 500 was in fact big enough to overcome the non-normality.

So my overall conclusion is that the *t*-test you did earlier was appropriate after all, even though you may not have expected it to be. It just goes to show how powerful the central limit theorem is when your samples really are large.

Energy expenditure

Do obese people use more energy on average than lean people? To find out, the total energy expenditure (in megajoules) was recorded over 24 hours for 22 people. 13 of these people were lean and 9 were obese. The data are shown in Figure 5, in dataframe `energy`. The energy expenditure is shown in column `expend`, and the classification of each person as lean or obese is shown in column `stature`.

(9) (2 points) A graph is shown in Figure 6. On the basis of this graph, and anything else you have learned about the data so far, explain briefly why any kind of two-sample *t*-test would not be appropriate.

We need both groups to have approximately a normal distribution given the sample size. The lean group has severe outliers, both lower and upper, so the sufficient normality fails. (Also, the obese group appears to have a right-skewed distribution, but you don't need to consider this once you've found one group that fails for sufficient normality. In any case, the sample size *might* be big enough for the obese people.)

Two points for finding *one* group that is not normal enough given its sample size. You don't need to say that *both* groups are not normal enough.

Strictly, you need to consider the sample size in *each* group, rather than the total, but I have already addressed that elsewhere (the income and education question), so I am willing to forgive it here. I ended up being a lot more relaxed about the marking of this one than I originally intended to be.

Extra: the inevitable bootstrapped sampling distributions of the sample means:

```
energy %>% nest_by(stature)
```

```
# A tibble: 2 x 2
  stature                data
  <fct>   <list<tibble[,1]>>
1 lean               [13 x 1]
2 obese               [9 x 1]
```

The two mini-data-frames in the list-column `data` have all the other columns that were in the original dataframe `energy`, that is to say, just `expend`, just for each of the people that were lean, and just for the ones that were obese. The `[,1]` under `data` is shorthand for "this has some number of rows (different for `lean` and `obese`), but only one column".

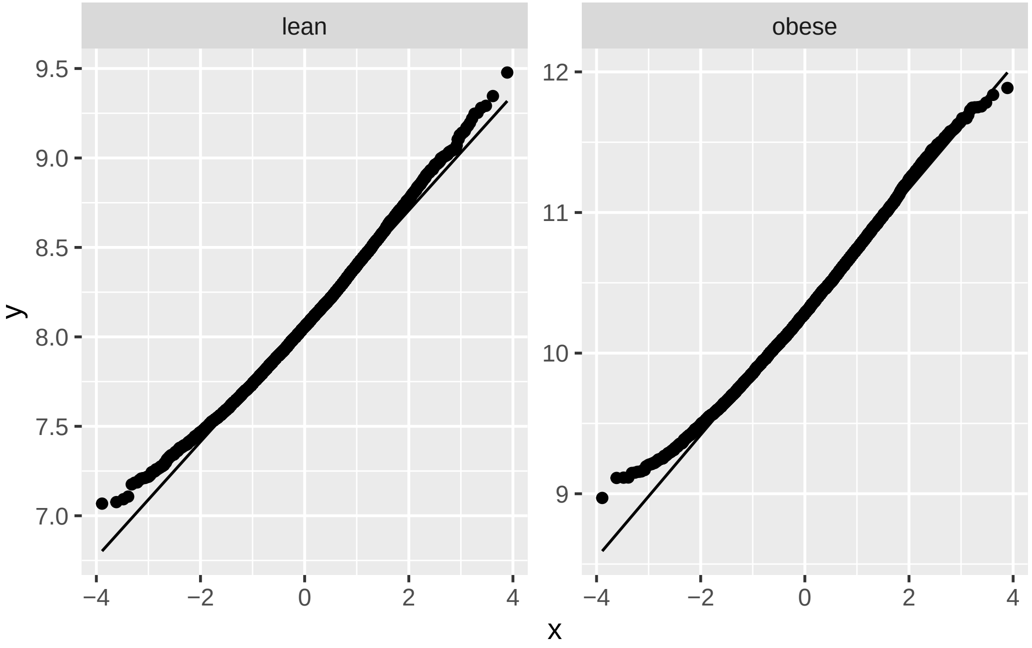
Now that we have the data we want, let's set up simulations:

```
energy %>% nest_by(stature) %>%
  mutate(sim = list(1:10000)) %>%
  unnest(sim)
```

```
# A tibble: 20,000 x 3
   stature                data   sim
   <fct>   <list<tibble[,1]>> <int>
 1 lean               [13 x 1]     1
 2 lean               [13 x 1]     2
 3 lean               [13 x 1]     3
 4 lean                 [13 x 1]     4
 5 lean                 [13 x 1]     5
 6 lean                 [13 x 1]     6
 7 lean                 [13 x 1]     7
 8 lean                 [13 x 1]     8
 9 lean                 [13 x 1]     9
10 lean                 [13 x 1]    10
# i 19,990 more rows
```

Now, we want a sample from each (from the `expend` in the appropriate `data`) and the mean of each of those samples, in familiar fashion:

```
energy %>% nest_by(stature) %>%
  mutate(sim = list(1:10000)) %>%
  unnest(sim) %>%
  rowwise() %>%
  mutate(my_sample = list(sample(data$expend, replace = TRUE))) %>%
  mutate(my_mean = mean(my_sample)) %>%
  ggplot(aes(sample = my_mean)) + stat_qq() + stat_qq_line() +
    facet_wrap(~stature, scales = "free")
```



The last two lines (or, one line broken into two parts) make normal quantile plots, separately for each group because each group is supposed to have a sampling distribution of the sample mean that is close to normal. I used `scales = "free"` because the typical energy usage of the two groups seems different. Both sets of points are pretty close to their lines, but:

- the `lean` group is (apparently very slightly) skewed to the right. This is because of those outliers, and to the right because there were two extreme outliers at the top and only one at the bottom.[1] This may not be as bad as you were expecting, with a sample size of only 13.
- the `obese` group is actually *not* skewed to the right, so that the sample size, even though only 9, *was* large enough to overcome that. What happened here is that the lower tail came out to be *short*, because the lower tail of the original distribution was also short. This is not in itself a problem for the t-test, but you only need one group to go wrong, and the `lean` group did.

(10) (2 points) What code would run a suitable test to see whether obese people use more energy on average than lean people? ("Average" can mean either mean or median, as appropriate.)

Mood's median test, having ruled out the Welch and pooled t-tests:

```
median_test(energy, expend, stature)
```

```
$grand_median
[1] 8.595

$table
       above
group   above below
  lean      2    11
  obese     9     0

$test
       what        value
1 statistic 1.523077e+01
2        df 1.000000e+00
3   P-value 9.514059e-05
```

[1] I suspect those three or so points at the bottom were when the bootstrap sample happened to contain three or so copies of the low outlier.

This is like the code for the sign test one on the midterm: there's nothing in the code that makes it one-sided. For the sign test, you get both one-sided P-values and the two-sided one, but Mood's median test is by definition *two*-sided, and we have some more work to do.

(11) (2 points) The output from the code you gave in the previous part is shown in Figure 7. What is an appropriate P-value for testing the alternative hypothesis of interest to us? Explain briefly.

This is output from a Mood's median test, and so is of course a clue about what test you need to be providing code for in the previous question. I expect you to recognize the test that this is output from.

We want to see whether obese people expend *more* energy on average than lean ones, so the alternative hypothesis of interest to us is *one*-sided. The P-value given in Figure 7 is *two*-sided (because Mood's median test is based on a chi-squared test, which is always two-sided), so we have to do two things:

- are we on the correct side?

  That is, are our data in favour of *our* alternative? If you look at the table of aboves and belows, all 9 of the obese people expended an above-average amount of energy, while only 2 of the 13 lean people did. This agrees with our alternative hypothesis that obese people expend more energy on average, so we are indeed on the correct side.

- If we are on the correct side, we need to divide the P-value by 2.

  Having established that we are on the correct side, our P-value is

```
9.514059e-05 / 2
```

```
[1] 4.75703e-05
```

or 0.000048.

The other, and you might say easier, way to establish that we are on the correct side is to look back at Figure 6 and note that the `obese` box is higher than the `lean` one: hence, that obese people do use more energy on average than lean ones, at least in the data we observed.

(12) (3 points) Hence, what do you conclude, in the context of the data?

"Hence" is a clue: it means "use the thing you just did to answer this question". (The word means "from here".)

We just worked out a one-sided P-value, 0.000048. This is smaller than 0.05. Our null hypothesis was that lean and obese people use the same amount of energy on average, and our (one-sided) alternative was that obese people use more energy. We reject the null in favour of the alternative, and thus conclude that obese people really do use more energy on average. (Or, "we have evidence that obese people use more energy on average").

State your P-value (the one from the previous part, even if you got that wrong), compare it with 0.05 (or similar), draw a conclusion about rejecting the null (or not), and then say what that means in the context of the data.

If you can get to the right conclusion by the right method without writing down your hypotheses, go for it, but it is safer to write down your hypotheses as well, so that we can see that you were drawing a conclusion consistent with them.

Extra: if you were wondering about how the t-test comes out, well, `lean` is before `obese` alphabetically, so:

```r
t.test(expend ~ stature, data = energy, alternative = "less")
```

```
    Welch Two Sample t-test

data:  expend by stature
t = -3.8555, df = 15.919, p-value = 0.0007053
alternative hypothesis: true difference in means between group lean and group obese is less
95 percent confidence interval:
      -Inf -1.220763
sample estimates:
 mean in group lean mean in group obese 
           8.066154           10.297778 
```

and the conclusion is actually about the same.

Extra: You are probably expecting to find now a confidence interval for the difference in medians, but there is no obvious way to do that. Mood's median test as we have done it tests only that the medians are the same (by computing the grand median and counting observations above and below that). To get a confidence interval, we would have to test whether the medians differed by a certain amount (and then include it in the confidence interval if that difference was not rejected, as for using the sign test to get a CI for a single median), which we do not have a good way to do. This question on Stack Overflow suggests a couple of ideas. One idea is based on the Hodges-Lehmann estimator, which is well-known but not quite the right thing for us.[2] Another is based on the bootstrap, which ought to be understandable to us: draw bootstrap samples from each group, take the difference in medians, and take the middle 95% of that distribution. The starting point is the same as I did before to assess the normality of the sampling distributions of the sample means:

```
energy %>% nest_by(stature)
```

```
# A tibble: 2 x 2
  stature               data
  <fct>   <list<tibble[,1]>>
1 lean              [13 x 1]
2 obese              [9 x 1]
```

with the list-column `data` containing two mini data frames, one for `lean` and one for `obese`, each with one column called `expend`. Then we set up 4 bootstrap samples[3] from each:

```
energy %>% nest_by(stature) %>%
  mutate(sim = list(1:4)) %>%
  unnest(sim) %>%
  rowwise() %>%
  mutate(my_sample = list(sample(data$expend, replace = TRUE)))
```

```
# A tibble: 8 x 4
  stature               data   sim my_sample
  <fct>   <list<tibble[,1]>> <int> <list>
1 lean              [13 x 1]     1 <dbl [13]>
2 lean              [13 x 1]     2 <dbl [13]>
3 lean              [13 x 1]     3 <dbl [13]>
4 lean              [13 x 1]     4 <dbl [13]>
5 obese              [9 x 1]     1 <dbl [9]>
6 obese              [9 x 1]     2 <dbl [9]>
7 obese              [9 x 1]     3 <dbl [9]>
8 obese              [9 x 1]     4 <dbl [9]>
```

[2] It uses the same idea of “what values would not be rejected by a test”, but from a different test, the rank sum test, which I think is inferior because it assumes the two distributions you are comparing have equal spread, in the same way that the pooled t-test does.

[3] To see whether it works. Once it does, we can scale it up.

Now we need the *medians* of each of those:

```
energy %>% nest_by(stature) %>%
  mutate(sim = list(1:4)) %>%
  unnest(sim) %>%
  rowwise() %>%
  mutate(my_sample = list(sample(data$expend, replace = TRUE))) %>%
  mutate(my_median = median(my_sample))
```

```
# A tibble: 8 x 5
  stature                data   sim my_sample  my_median
  <fct>   <list<tibble[,1]>> <int> <list>         <dbl>
1 lean               [13 x 1]     1 <dbl [13]>      8.11
2 lean               [13 x 1]     2 <dbl [13]>      7.48
3 lean               [13 x 1]     3 <dbl [13]>      8.08
4 lean               [13 x 1]     4 <dbl [13]>      7.9
5 obese               [9 x 1]     1 <dbl [9]>       9.21
6 obese               [9 x 1]     2 <dbl [9]>       9.68
7 obese               [9 x 1]     3 <dbl [9]>       9.19
8 obese               [9 x 1]     4 <dbl [9]>       9.21
```

Here is where things turn different: we want to put the medians for the *same* simulation next to each other and subtract them, to get a bootstrap version of what the difference in medians might look like. This is an unexpected use of `group_by` and `summarize`:

```
energy %>% nest_by(stature) %>%
  mutate(sim = list(1:4)) %>%
  unnest(sim) %>%
  rowwise() %>%
  mutate(my_sample = list(sample(data$expend, replace = TRUE))) %>%
  mutate(my_median = median(my_sample)) %>%
  group_by(sim) %>%
  summarize(med1 = my_median[1], med2 = my_median[2])
```

```
# A tibble: 4 x 3
    sim  med1  med2
  <int> <dbl> <dbl>
1     1  8.11  9.21
2     2  7.48  9.68
```

```
3      3  8.08  9.19
4      4  7.9   9.21
```

We group by simulation (to keep things from the same simulation together) and then pull out the first median that belongs to each simulation (the one from the lean people) and then the second one (from the obese people). Square brackets are used to grab a particular median within a group.

Then we take the difference of the two medians within each sim, and pull out the 2.5 and 97.5 percentiles of the whole distribution of the differences (so that 95% of the bootstrap distribution is between them):

```
energy %>% nest_by(stature) %>%
  mutate(sim = list(1:4)) %>%
  unnest(sim) %>%
  rowwise() %>%
  mutate(my_sample = list(sample(data$expend, replace = TRUE))) %>%
  mutate(my_median = median(my_sample)) %>%
  group_by(sim) %>%
  summarize(med1 = my_median[1], med2 = my_median[2]) %>%
  mutate(diff = med1 - med2) %>%
  summarize(lwr = quantile(diff, 0.025), upr = quantile(diff, 0.975))
```

```
# A tibble: 1 x 2
    lwr   upr
  <dbl> <dbl>
1 -2.13 -1.10
```

There is our proof of concept, even though it is kind of dopey with only four simulations. So let's up the number of simulations to 10,000 (we are using the extremes of the distribution, so we need more simulations to estimate them accurately):

```
energy %>% nest_by(stature) %>%
  mutate(sim = list(1:10000)) %>%
  unnest(sim) %>%
  rowwise() %>%
  mutate(my_sample = list(sample(data$expend, replace = TRUE))) %>%
  mutate(my_median = median(my_sample)) %>%
  group_by(sim) %>%
  summarize(med1 = my_median[1], med2 = my_median[2]) %>%
  mutate(diff = med1 - med2) %>%
  summarize(lwr = quantile(diff, 0.025), upr = quantile(diff, 0.975))
```

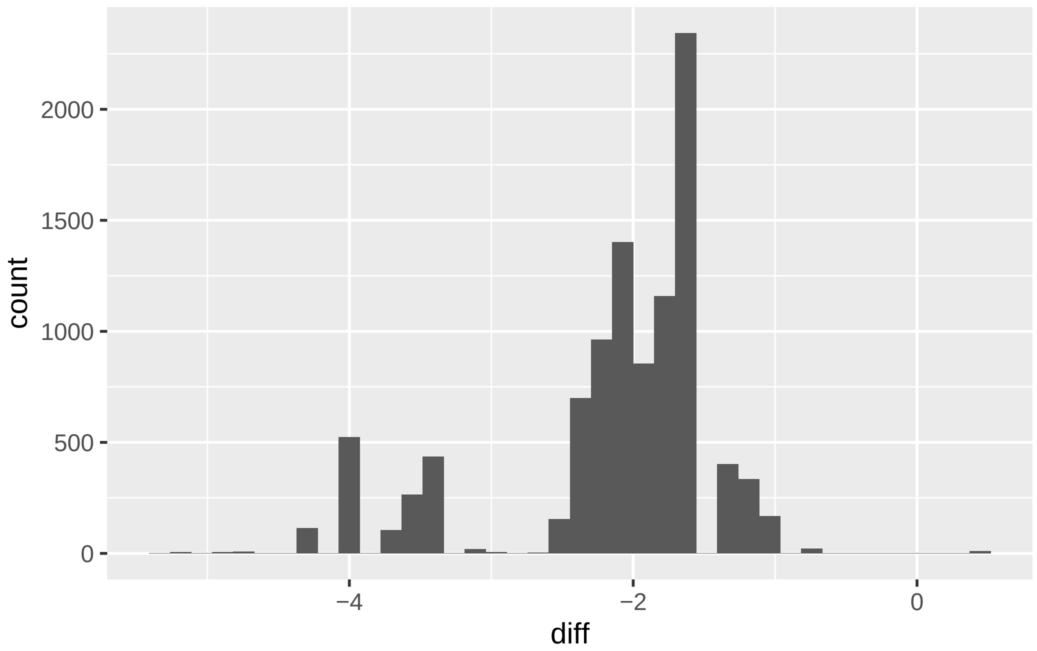
```
# A tibble: 1 x 2
    lwr   upr
  <dbl> <dbl>
1 -3.98 -1.12
```

and that, after some considerable effort, is a 95% CI for the difference in medians. If you can remember back to when we did our test, you'll recall that we concluded that obese people do expend more energy on average than lean ones. This interval is lean minus obese, so it makes perfect sense that the interval is entirely negative: obese people expend between 1.12 and 3.98 megajoules more than lean people on average, over 24 hours.

This interval indicates the usual thing: we are sure that there is a difference between lean and obese people (small P-value), but we are unsure about how big that difference is (longish confidence interval relative to zero). The remedy is the usual one: take sample sizes bigger than the rather tiny ones here.

Extra extra: There is usually something funny about bootstrapping medians. Let's take a look at the distribution of the column of differences, via a histogram:

```
energy %>% nest_by(stature) %>%
  mutate(sim = list(1:10000)) %>%
  unnest(sim) %>%
  rowwise() %>%
  mutate(my_sample = list(sample(data$expend, replace = TRUE))) %>%
  mutate(my_median = median(my_sample)) %>%
  group_by(sim) %>%
  summarize(med1 = my_median[1], med2 = my_median[2]) %>%
  mutate(diff = med1 - med2) %>%
  ggplot(aes(x = diff)) + geom_histogram(bins = 40)
```



I've used more bins than I usually would to show you that this is not close to a normal distribution, or even any kind of regular unimodal one.[4] The problem is that the numerical values in a bootstrap sample are the same as the ones in the original sample (with possibly more or less of each) and the median must be one of those (or possibly halfway between them if the sample size is even). So there are not many possible values for the median of a bootstrap sample, and thus not very many possible values for the difference between the sample medians, and therefore you get this kind of "clumpy" bootstap distribution. As far as getting a confidence interval is concerned, this is not really a problem (you just take the appropriate percentiles, or in more advanced work with the bootstrap, you adjust things a bit, such as to use a "bias-corrected and accelerated" interval.[5]).

Income and education

These data are annual incomes in 2005 of a random sample of 2,584 Americans who were selected for the National Longitudinal Survey of Youth in 1979 and who had paying jobs in 2005. The columns are:

- `Subject`: a subject ID

[4] If we had done this for the *mean*, the distribution would probably have been a bit skewed, but would have had one clear peak and a nice smooth shape.

[5] The wisdom is that the percentile-based interval we got will generally be too short.

- Educ: the amount of education (see below)
- Income2005: annual income in 2005
- tr_income: transformed 2005 income (which we will use in our analysis).

The categories in Educ are:

- 012 (less than high school, 0–12 years)
- 12 (completed high school, 12 years)
- 13-15 (some college or university, 13–15 years)
- 16 (completed university, 16 years)
- 16+ (started or completed graduate school, more than 16 years).

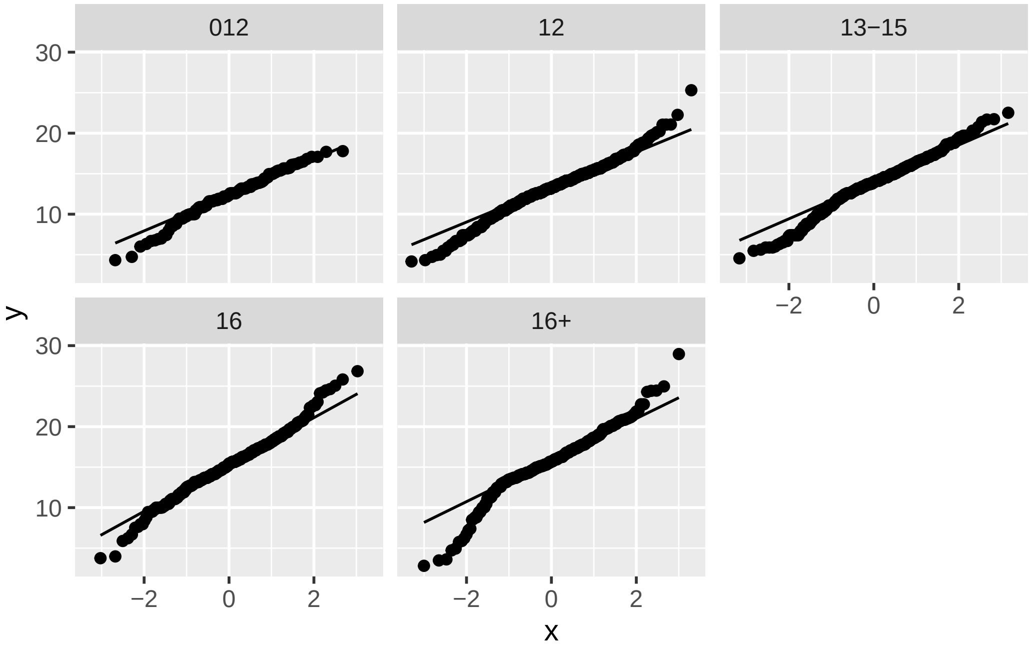
The people who collected the data expected to see that more education went with a higher income on average (or a higher transformed income).

Some of the data, in dataframe incomes, is shown in Figure 8. Summary data of transformed income for each education category are shown in Figure 9.

(13) (3 points) Plots of the transformed incomes are shown in Figure 10. What code was used to make this Figure?

These are normal quantile plots of transformed income separately for each education category, that is, facetted by Educ. (Saying this can help you get partial credit even if your code is messed up.) Hence:

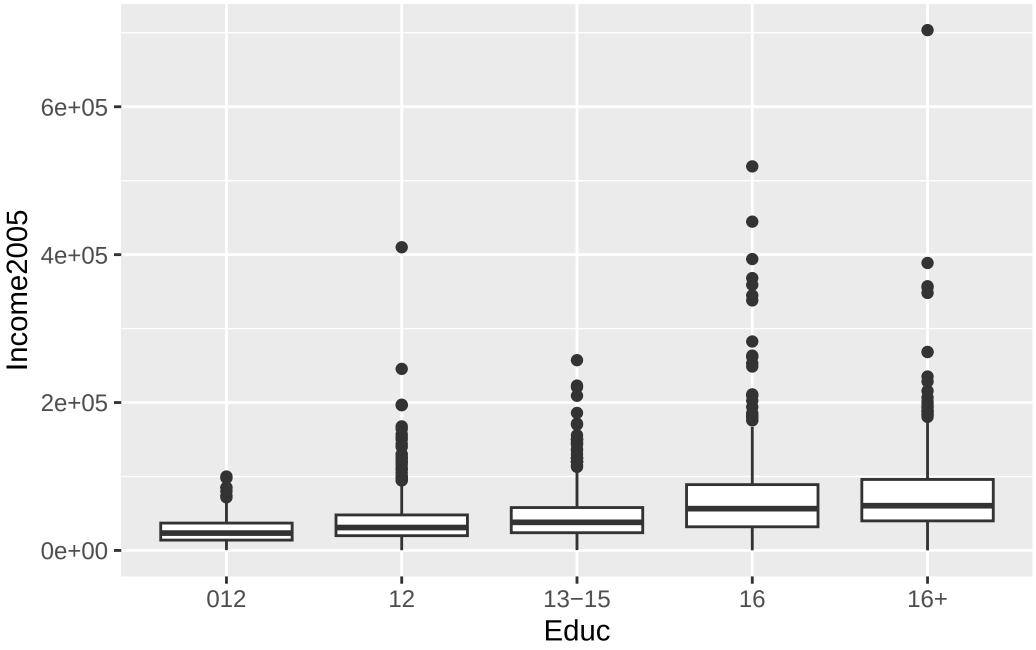
```
ggplot(incomes, aes(sample = tr_income)) + stat_qq() + stat_qq_line() +
  facet_wrap(~ Educ)
```



Two points for a normal quantile plot of the transformed incomes (minus one if you use the *untransformed* incomes), and the last point for the correct `facet_wrap`. 0.5 of the last point if you do a `facet_wrap` but somehow mess it up, for example if you add a `scales = "free"`. Each of the five normal quantile plots is on the same scale, so there is no `scales = "free"`.

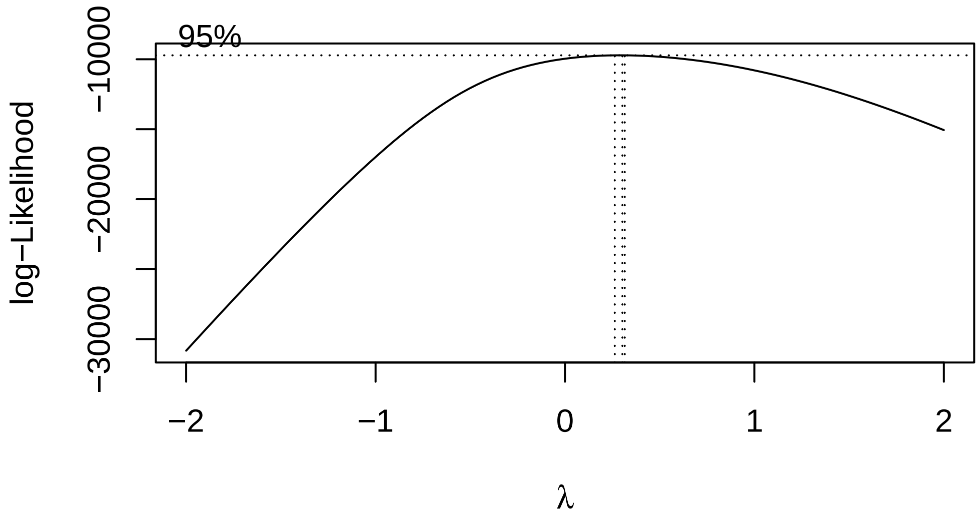
Extra: if you are curious about the transformed incomes, here is a boxplot of the *untransformed* incomes:

```
ggplot(incomes, aes(x = Educ, y = Income2005)) + geom_boxplot()
```



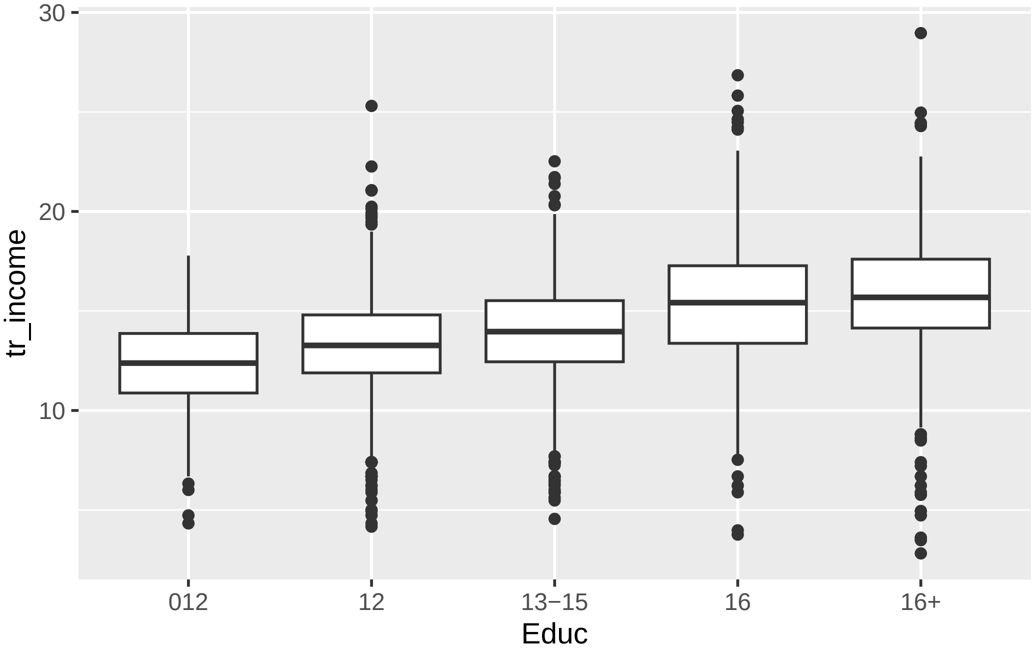
These are, evidently, *very* right-skewed, as is very common for incomes. You also see that when the median income is bigger (for the more highly-educated groups), the spread of incomes is also bigger. This is the kind of situation in which a transformation is worth trying. I was expecting a log-transform to be the thing, but I ran Box-Cox (as you can also do for ANOVA), and got this:

```
library(MASS, exclude = "select")
boxcox(Income2005 ~ Educ, data = incomes)
```



The confidence interval for λ is very narrow, because we have so much data, and definitely does not include zero (log) or 0.5 (square root). I figured the best transformation was something like power 0.25 (fourth root), which is not exactly common (so I didn't ask you about it), which is what `tr_income` actually is:

```
ggplot(incomes, aes(x = Educ, y = tr_income)) + geom_boxplot()
```



These distributions are at least more or less symmetric, though they seem to have a lot of outliers. The problem with boxplots when you have a lot of data, though, is that they will appear to exaggerate the number of outliers. Thus, I gave you normal quantile plots rather than this boxplot, so as to confuse you less.

(14) (2 points) According to Figure 10, what do the distributions of transformed incomes appear to have in common (that is, what is the same about them)? Why, nonetheless, is it *not* necessary to run Mood's median test here to compare the transformed incomes, based on the information you have?

The overall picture appears to be that these distributions are long-tailed: they go both up and down further than you would expect from a normal distribution. One point. "They have outliers" is not precise enough (or even accurate); "they have upper and lower outliers" is closer to the mark, but the highest and lowest values are mostly not clearly higher or lower than the other values, so that the length of tail is a feature of the whole distribution. Even if you think there are outliers in some of these (say, at the upper end of `16+`), there are a lot more points clearly off the line than just the most extreme ones. So, nothing for "outliers" and a rather generous 0.5 for "upper and lower outliers".

The other information you have is the sample sizes in Figure 9. These are all huge (even the smallest sample is over 100), so we will get a *lot* of help from the Central Limit Theorem. The long tails are (relatively speaking) not too bad, and these huge sample sizes are large enough to take care of them. Hence the normality is good enough, and we have no need to go to Mood's median test. The second point. (The "nonetheless" is a bit of a clue that the answer will involve something else that you haven't considered yet, such as sample sizes.) Half a point for "all the sample sizes are large", and the other half point for why it helps. "The sample size is large" is not precise enough: *which* sample size? There are five of them, and the total of the five sample sizes is not the relevant consideration for whether things are normal enough. (On this last point, if the sample sizes were all in the hundreds except for one that is 5, and that distribution is obviously not normal, then we have a problem, regardless of the fact that the total sample size is several hundred.)

I think you need to say something more than "the distributions are approximately normal", but even if that is your conclusion, you really need to say something about sample sizes: sample sizes of, say, $n = 5$ might mean that you ought to ask for something closer to normality than you see here.

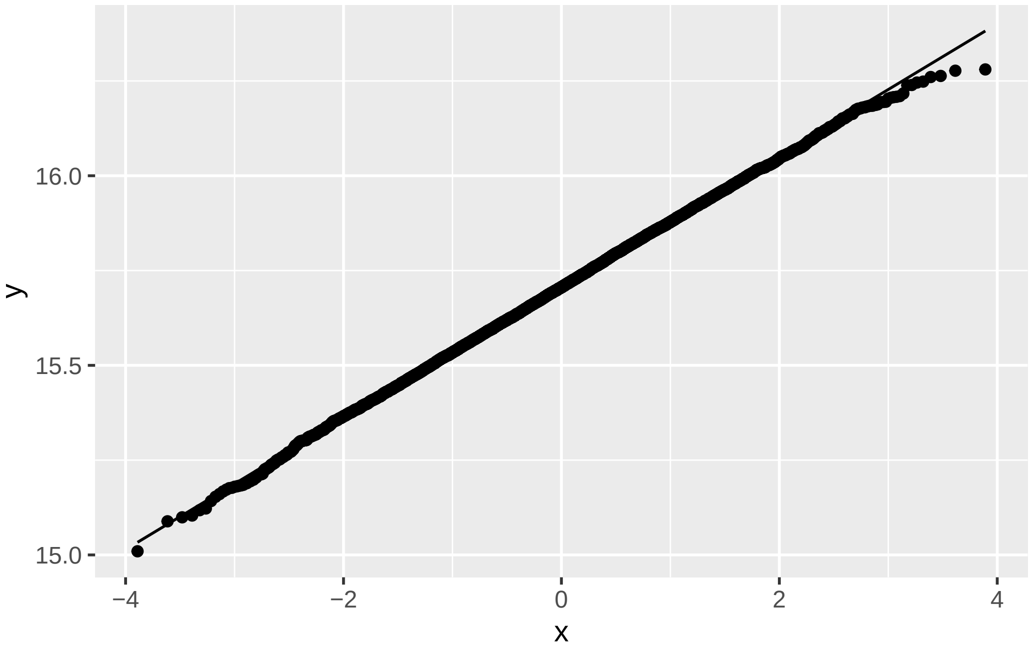
Extra: if you don't believe me about the actual sample sizes being big enough, pick what looks like the worst one, and get hold of a bootstrap sampling distribution of the sample mean. I think the worst one is the `16+` group; the one with the smallest sample size, `012`, is actually close to normal, so there is no worry about that one. Hence, grab the `16+` Educ people, and:

```r
incomes %>%
  filter(Educ == "16+") -> g16
g16
```

```
# A tibble: 374 x 4
   Subject Educ  Income2005 tr_income
     <int> <chr>      <int>     <dbl>
 1      20 16+        64000      15.9
 2      46 16+        42000      14.3
 3      58 16+        55500      15.3
 4      69 16+       120000      18.6
 5      73 16+       186135      20.8
 6     121 16+        41000      14.2
 7     140 16+        65000      16.0
 8     141 16+        44768      14.5
 9     158 16+        35100      13.7
10     162 16+        76000      16.6
# i 364 more rows
```

and then

```
tibble(sim = 1:10000) %>%
  rowwise() %>%
  mutate(my_sample = list(sample(g16$tr_income, replace = TRUE))) %>%
  mutate(my_mean = mean(my_sample)) %>%
  ggplot(aes(sample = my_mean)) + stat_qq() + stat_qq_line()
```



As normal as you could ever want. The long-tailedness has all but gone away.

(15) (3 points) Two analyses are shown in Figure 11 and Figure 12. Which analysis is better? Explain briefly. Your answer should make it clear what each analysis is.

These are a regular ANOVA and a Welch ANOVA (as you can tell from the output). One rather easy point for recognizing this.

The decision between them is based on whether you think the spreads (of transformed incomes within groups) are approximately equal.

You have two options to assess this:

- look at the standard deviations in Figure 9, and make a call about whether these are close to equal. I think they are.
- look at the normal quantile plots in Figure 10 and makes a decision based on the *slopes* of the lines there. I don't see much difference.

One point for considering equality of spreads based on the information you have.

So I would go with the regular ANOVA in Figure 11. I have no objection if you disagree with me, as long as you do so for a good reason. If, for example, you think the SDs in Figure 9 are (sufficiently) different, you need to prefer the Welch ANOVA. The third point for a preferred analysis consistent with what you said about spreads.

It is very important to learn to answer these kinds of questions *concisely*: that is, to answer the question as asked without rambling. In an exam, writing too much means that the grader may not be able to find the right answer in amongst the irrelevance (bear in mind that the grader has no more than thirty seconds to read and assess your answer to a question like this, so it is in your interest to make your answer *as easy as possible* to grade). In real life, it means that your reader does not understand and is unable to act upon what you are saying because you wrote too much that they do not care about.

(16) (3 points) What conclusions do you draw from your chosen analysis, bearing in mind what the people who collected the data would like to know?

The people who collected the data suspected that more education would go with more income, so that is what we need to address in the end.

One step at a time, though. The conclusions are exactly the same whichever analysis you preferred, so:

- the F-test says that there are some differences in mean (transformed) income (or that the mean transformed income is not the same for all levels of education). That's as far as that goes. We proceed to the followup analysis (Tukey for `aov`, Games-Howell for the Welch ANOVA).
- Mean (transformed) income is different for each pair of education categories except for `16` (completed university) and `16+` (at least some graduate school). If you look back at Figure 9, mean income goes steadily up with education, but the smallest difference is between the last two education categories.

  So what you say is something like "these data show that income does indeed increase with the amount of education, except that there is no significant increase in income between completing university and attending graduate school".

Marks: one for a proper conclusion from the F-test, one more for some discussion of the followup results, one more for something sufficiently close to my last sentence that talks about the relationship between income and education. My comments from the previous question about conciseness apply here too. If you write too much, do not be surprised if the grader stops reading at some point.

Extra 1: you might have noticed that Figure 11 and Figure 12 had identical results, and, indeed, very similar P-values in each case, so that it didn't matter which one we did. This (as here) is usually a sign that the spreads were close enough to equal for `aov` to be reasonable.

Extra 2: in case you were curious, Mood's median test gives very similar results again:

```
library(smmr)
median_test(incomes, tr_income, Educ)
```

```
$grand_median
[1] 13.98311

$table
       above
group   above below
  012      30   106
  12      383   637
  13-15   320   328
  16      271   135
  16+     288    86

$test
       what        value
1 statistic 2.604786e+02
2        df 4.000000e+00
3   P-value 3.596309e-55
```

```
pairwise_median_test(incomes, tr_income, Educ)
```

```
# A tibble: 10 x 4
   g1    g2     p_value adj_p_value
   <chr> <chr>    <dbl>       <dbl>
 1 012   12    6.96e- 4    6.96e- 3
 2 012   13-15 9.93e- 9    9.93e- 8
 3 012   16    8.62e-17    8.62e-16
 4 012   16+   7.09e-21    7.09e-20
 5 12    13-15 5.58e- 7    5.58e- 6
 6 12    16    7.11e-22    7.11e-21
 7 12    16+   1.07e-37    1.07e-36
 8 13-15 16    2.95e- 9    2.95e- 8
 9 13-15 16+   4.60e-19    4.60e-18
10 16    16+   2.20e- 1    1   e+ 0
```

The P-values are a bit bigger than for the other two ANOVAs, but the conclusions are identical. This is another sign that it doesn't matter what test you do, so you might as well use the most powerful test (`aov`) since the normality seems OK. (If the P-values had come out more dissimilar, that would have required us to make a decision about which one we believed.)

Another thing: since Mood's median test doesn't make any assumptions about normality, what if we had done it on the *untransformed* incomes?

```
library(smmr)
median_test(incomes, Income2005, Educ)
```

```
$grand_median
[1] 38231

$table
       above
group   above below
  012      30   106
  12      383   637
  13-15   320   328
  16      271   135
  16+     288    86

$test
       what        value
1 statistic 2.604786e+02
2        df 4.000000e+00
3   P-value 3.596309e-55
```

```
pairwise_median_test(incomes, Income2005, Educ)
```

```
# A tibble: 10 x 4
   g1    g2     p_value adj_p_value
   <chr> <chr>    <dbl>       <dbl>
```

```
 1 012   12    6.96e- 4    6.96e- 3
 2 012   13-15 9.93e- 9    9.93e- 8
 3 012   16    8.62e-17    8.62e-16
 4 012   16+   7.09e-21    7.09e-20
 5 12    13-15 5.58e- 7    5.58e- 6
 6 12    16    7.11e-22    7.11e-21
 7 12    16+   1.07e-37    1.07e-36
 8 13-15 16    2.95e- 9    2.95e- 8
 9 13-15 16+   4.60e-19    4.60e-18
10 16    16+   2.20e- 1    1   e+ 0
```

Identical!

All the transformations we consider are “monotonic”, meaning that if one untransformed value is larger than another, the first *transformed* value is larger than the second one as well. The impact that has on Mood’s median test is that the median of the transformed values is the transformed value of the median of the original values[6] (the untransformed and transformed values are in the same order), *and* an untransformed value that is above its grand median will *also* have a transformed value that is above its grand median. That is to say, a monotonic transformation will not change the counts of values above and below, and so will not change any of the P-values in Mood’s median test.

Extra 3: this was an observational study rather than an experiment, so we cannot say that a larger amount of education *causes* a larger income. Undoubtedly it does to a certain degree, but things like parental wealth (or education, or attitudes towards education) will also have an impact. For example, if your parents are rich, you might be able to get into Harvard and make a lot of money afterwards, but that came mainly from your parents and the connections you made at Harvard, not (or not completely) from the fact that you have four more years of education.

Tidying

The following questions are about tidying data.

(17) (3 points) Figure 13 shows a dataframe `d1` and, below it, some code run with this dataframe. What will be the *output* of this code?

This:

[6] You may need to read that a second time. The idea is that it doesn’t matter whether you transform (all the values) first and then take the median, or take the median first and then transform it.

```
d1 %>% pivot_longer(a:b, names_to = "trt", values_to = "yield")
```

```
# A tibble: 6 x 3
      r trt   yield
  <dbl> <chr> <dbl>
1     1 a        12
2     1 b        14
3     2 a        13
4     2 b        15
5     3 a        16
6     3 b        17
```

This is a standard pivot-longer. There is a new column `trt` that contains the values `a` and `b` (`trt` is short for "treatment") and a new column `yield` that contains the numbers in those two columns, matched up with their treatment ("yield" is meant to suggest a response variable). The column `r` (short for "replicate") is not named in the `pivot_longer`, so it will get repeated as necessary to match the right treatment and yield.

When you're doing this, you can do a fill-in-the-blank: you know there are going to be new colums `trt` and `yield`, the columns `a` and `b` are going away, and the other column not mentioned, `r`, will still be in the result. You also know there are going to be 6 rows, and your job is to make sure that the values of `r`, `trt`, and `yield` still match up (for example, the values 12, 13, and 16 for `yield` match up with `a`, and with values of `r` of 1, 2, and 3 respectively).

Three points for getting this exactly, or something equivalent to it that matches up the right values (for example, all the `a` first in `trt`, matched up with `r`-values of 1, 2, and 3 and `yield` values of 12, 13, and 16.) 2.5 for this with a small error, in the grader's judgement. Two points for getting "more than halfway" to the right answer (in the grader's judgement), and one for showing some progress towards the right answer.

I realize that the grader will need to be careful here (see my very last sentence for a grading tip), since there are several possible right answers. Here is the one I mentioned above, which is also three points:

```
# A tibble: 6 x 3
      r trt   yield
  <dbl> <chr> <dbl>
1     1 a        12
2     2 a        13
```

```
3     3 a         16
4     1 b         14
5     2 b         15
6     3 b         17
```

and this, though weird, is also correct, since the right things go together:

```
# A tibble: 6 x 3
      r trt   yield
  <dbl> <chr> <dbl>
1     1 a        12
2     2 a        13
3     1 b        14
4     2 b        15
5     3 a        16
6     3 b        17
```

I am not expecting you to know which order the rows are in, or for that matter the columns. If you have some way of keeping the right values together, then you are good. Expect the grader to check that the `yield` value of 12, 13, and 16 go with `trt a` and with `r` of 1, 2, and 3 in that order, regardless of the arrangement of rows and columns. (This was how I actually checked it.)

A few people tried to describe the *form* of the output, but the reason I italicized "output" in the question is that you have enough information to work out what the *actual output* is. I couldn't award more than one point for this, because it didn't show that you knew which values would go into which places (and that was if your description was accurate enough).

(18) (4 points) Figure 14 shows a dataframe `d2` and Figure 15 shows a dataframe `d3`. What code will rearrange `d2` into `d3`? For full credit, do this in one step.

This is what I actually did:

```
d2 %>% pivot_longer(everything(), names_to = c(".value", "time"),
                    names_sep = "_") -> d3
d3
```

```
# A tibble: 4 x 3
  time      x     y
  <chr> <dbl> <dbl>
1 1        21    11
2 2        22    12
3 1        23    13
4 2        24    14
```

The thing to recognize is that part of the current column names is used to make the new column names `x` and `y`, so we need to use the special `.value` to make that happen in one step. The rest of the old column names (the 1 and the 2) go into the new column `time`, so `time` is named as normal in the `names_to` as well. The two bits of the old column names are separated by a `_`, which is specified in `names_sep`. Four points.

In case you are wondering where the `values_to` went: this is taken over by the `.value` thing: the names of the new columns that contain the values are taken from the appropriate bit of the old column names, rather than being supplied by us. (If you supply one, your thought process ought to be "well, what is this going to be used for and what value do I put in it?", and then you realize that you don't have one.)

Some people tried to put two things into `values_to`, like this:

```
d2 %>% pivot_longer(everything(), names_to = "time", names_sep = "_",
                    values_to = c("x", "y"))
```

```
Error in `pivot_longer()`:
! `names_sep` can't be used with a length 1 `names_to`.
```

This doesn't work because it hasn't said *which* of the numeric values belong in `x` and which in `y`. In general, you can only have *one* thing in `values_to` at most. This sort of thing is only one point.

Another one I saw was two pivots-longer:

```
d2 %>% pivot_longer(x_1:x_2, names_to = "time", values_to = "x") %>%
  pivot_longer(y_1:y_2, names_to = "time2", values_to = "y")
```

```
# A tibble: 8 x 4
  time      x time2     y
  <chr> <dbl> <chr> <dbl>
1 x_1      21 y_1      11
2 x_1      21 y_2      12
```

```
3 x_2      22 y_1      11
4 x_2      22 y_2      12
5 x_1      23 y_1      13
6 x_1      23 y_2      14
7 x_2      24 y_1      13
8 x_2      24 y_2      14
```

which is not insane, but now we have values repeated and we'd need to "stack" the columns (and you have to give your second `names_to` a different name). Two if you used a different name for your second `names_to`, 1.5 if you used the same one.

If you don't see the `.value` thing, you might think of something like this:

```
d2 %>% pivot_longer(everything(), names_to = c("xy", "time"),
                    names_sep = "_", values_to = "xy_value")
```

```
# A tibble: 8 x 3
  xy    time  xy_value
  <chr> <chr>    <dbl>
1 x     1           21
2 x     2           22
3 y     1           11
4 y     2           12
5 x     1           23
6 x     2           24
7 y     1           13
8 y     2           14
```

but this is now too long (and you might have trouble thinking of names for those things). You can recognize that this will be too long because there are eight values to go into what I called `xy_value`, and each one of them will be on its own row. One point for getting this far and stopping.

So now you have to make this wider: make columns whose names are in `xy` and whose values are in `xy_value`:

```
d2 %>% pivot_longer(everything(), names_to = c("xy", "time"),
                    names_sep = "_", values_to = "xy_value") %>%
  pivot_wider(names_from = xy, values_from = xy_value)
```

```
# A tibble: 2 x 3
  time  x         y
  <chr> <list>    <list>
1 1     <dbl [2]> <dbl [2]>
2 2     <dbl [2]> <dbl [2]>
```

This doesn't work because there are only two distinct values of `time`, and so there are only two rows doing it this way (and `d3` has four rows). Two points for getting this far, and one point for saying that this won't be right (and something about why).

This is another way to three points:

```
d2 %>% pivot_longer(everything(), names_to = c("xy", "time"), names_sep = "_",
                    values_to = "xy_value") %>%
  pivot_wider(names_from = xy, values_from = xy_value) %>%
  unnest_longer(c(x, y))
```

```
# A tibble: 4 x 3
  time      x     y
  <chr> <dbl> <dbl>
1 1        21    11
2 1        23    13
3 2        22    12
4 2        24    14
```

Or this, using `unnest` on the last line:

```
d2 %>% pivot_longer(everything(), names_to = c("xy", "time"), names_sep = "_",
                    values_to = "xy_value") %>%
  pivot_wider(names_from = xy, values_from = xy_value) %>%
  unnest(c(x, y))
```

```
# A tibble: 4 x 3
  time      x     y
  <chr> <dbl> <dbl>
1 1        21    11
2 1        23    13
3 2        22    12
4 2        24    14
```

Another approach that will get you something if you do it right is this:

```
d2 %>% pivot_longer(everything(), names_to = "thing", values_to = "z") %>%
  separate_wider_delim(thing, names = c("xy", "time"), delim = "_") %>%
  pivot_wider(names_from = xy, values_from = z)
```

```
# A tibble: 2 x 3
  time  x          y
  <chr> <list>     <list>
1 1     <dbl [2]> <dbl [2]>
2 2     <dbl [2]> <dbl [2]>
```

One point for this, and a second for explaining what will happen or for trying an `unnest`.

Minus a half point for small errors.

For something as potentially complicated as this, you stand to gain a lot by *explaining* your process, and not just writing down some apparently random code, because then, even if you are wrong, the grader can award you some credit for a sensible thought process.

(19) (3 points) Dataframe `d4` is shown in Figure 16, and dataframe `d5` is shown in Figure 17. What code will rearrange `d4` into `d5`? Explain briefly how you know your code will work (hint: how do you know that the rows will be correct?).

This is a straightforward `pivot_wider`, so just this:

```
d4 %>% pivot_wider(names_from = g, values_from = x) -> d5
d5
```

```
# A tibble: 3 x 3
  rep    trt1  trt2
  <chr> <dbl> <dbl>
1 r1       10    13
2 r2       11    14
3 r3       12    15
```

Two points.

You know that this will work, because the thing not named in the `pivot_wider`, namely `rep`, will determine what rows things go into (so that there will be three rows labelled `r1` through `r3`, as `d5` has). The other point.

(20) (3 points) Figure 18 shows a dataframe `d6` and some code that uses `d6`. What will the *output* from this code be?

This:

```
d6 %>% pivot_wider(names_from = v, values_from = w)
```

```
# A tibble: 3 x 3
  u        g1    g2
  <chr> <dbl> <dbl>
1 a        10    NA
2 b        14    11
3 c        NA    13
```

This is one of the not-enough-data ones. A hint for these is to create an empty dataframe with the columns you know it will have (the values in `v`) and the rows you know it will have (the values in `u` that was not mentioned in the pivot-wider), and put the values in `w` in the right places according to their corresponding values for `u` and `v`. At the end, you'll have two empty cells; put NA in those and you're done.

The rows and/or columns can be in a different order than shown (since you won't know for sure which order they'll come out, but alphabetical order seems like a good guess). What matters is that each of the values comes out in the row with its value of `u` and the column with the right value from `v`, with missings where there are no values to go.

The grader will award you a mark according to how close they thought you got to the above: three points for exactly that, two points for the right structure but some wrong values, one point for some sort of sensible idea even if the answer was wrong, etc.

Growing cattle

As cattle (cows and bulls) grow, the metabolic clearance rate of growth hormone changes. In a study, 14 male cattle were weighed, and their metabolic clearance rate was measured. The data are shown in Figure 19, with the metabolic clearance rates in column `mcr`. The dataframe is called `growth`.

(21) (3 points) A scatterplot of the data is shown in Figure 20. Describe any trend you see (hint: form, direction, strength).

I see:

- form: a upward-opening curve shape (not linear; “a curved shape” is enough). It is, to my mind, definitely not linear; if it were, those points at a body weight near 500 would have much lower values of `mcr`.
- direction: downward trend
- strength: a moderate trend at best (there is a fair bit of scatter among `mcr` values at the same `weight`, or the points are not especially close to whatever trend there is).

A point each for mentioning these. For the strength one, if you think the relationship is “strong”, I will need to know why you think that (minus a half if you don’t say). I actually disagree with “strong” as a description, but if you can make the case for it, I am happy that you earned the point.

If you can add a (brief!) justification, that strengthens your answer, but I realize that I didn’t ask for it, so a clear description that I can go along with is enough.

This is an ordinary scatterplot, so nothing here is supposed to have a “normal distribution” (for that, you would look at something like a normal quantile plot of the residuals).

(22) (2 points) Some analysis is shown in Figure 21 and Figure 22. Which part of this analysis tells you that that it might be a good idea to add a squared term in body weight to the regression, and how does it tell you this?

Figure 22 (the plot of residuals against fitted values). One point for this. This shows a curved pattern (down and then up again), indicating that the relationship between `mcr` and body weight is curved and would be better modelled by adding a squared term in body weight, rather than a random pattern as would be desirable (the other point for these).

Make sure you’re clear about what this plot is: it is not a scatterplot (that is Figure 20), but a plot of the residuals on which you were hoping for *no* pattern, and the existence of the curved pattern is the reason for the modification of the model.

Extra: in a “simple” regression, with only one x-variable, you cannot really disentangle transforming the explanatory side (by adding x-squared) or transforming the response side (for example, by taking logs, or using Box-Cox), because the plot of residuals against fitted is all you have to work with. Hence my framing of this question: I told you what I was thinking of doing, and it was up to you to come up with a reason why I might do it. (If I had asked you, on the basis of Figure 22, “what would you do next?”, the right answer could be either to add the square of body weight or to try a transformation of `mcr`; either of these would have been an acceptable thing to try.) In a multiple regression (which this is not), you have more graphs to work with, and it is then clearer about what to do, depending on which of the graphs you have a problem with.

(23) (2 points) A regression that includes body weight squared is shown in Figure 23. What are *two* ways you can tell that adding the squared term was a good idea?

These:

- the squared term is significant (P-value 0.0218), indicating that it adds to the fit of the regression over and above the linear term.
- R-squared has increased substantially, from 0.697 to 0.816. (A word like "substantially" needs to be there, because R-squared will increase a little even if the squared term is worthless.)

A point for each. Only 0.5 for saying that R-squared increases without saying something about the change being large, or something equivalent.

Extra: When you have repeated observations at the same x, as we do here (three or four different observations at the same `body_weight`), you can figure out whether the "error" in the regression is due to random error or to something with the model that needs to be improved. Here, there seems to be quite a bit of random error because the `mcr` can differ substantially even among cattle of the same weight. A way of doing this is to subdivide the error sum of squares into a part that is due to "pure error" (dissimilar `mcr` values among cattle of the same `body_weight`), and the rest of it, which is called "lack of fit".

A function that does this is called `pureErrorAnova` from the `alr3` package.[7] The models we fitted are called `growth.1` (linear model from Figure 21) and `growth.2` (quadratic model from Figure 23). Let's try the first of these:[8]

```
library(alr3)
pureErrorAnova(growth.1) %>% knitr::kable()
```

	Df	Sum Sq	Mean Sq	F value	Pr(>F)
body_weight	1	16634.201	16634.2011	38.14309	0.0001047
Residuals	12	7241.013	603.4178	NA	NA
Lack of fit	2	2880.013	1440.0066	3.30201	0.0792380
Pure Error	10	4361.000	436.1000	NA	NA

A decent fraction of the error sum of squares from this regression is due to lack of fit (the rest of it is pure error), and the lack of fit is close to significance (P-value 0.079, which is small if not quite smaller than 0.05). This hints at the idea that we can improve our model, which we did by adding the squared term to get model `growth.2`:

[7] When I tried to install this the usual way, I was told that it no longer existed on CRAN, but I found a way around this. Ask me if you want to know about it.

[8] The `knitr::kable` makes a nicer table.

```
pureErrorAnova(growth.2) %>% knitr::kable()
```

	Df	Sum Sq	Mean Sq	F value	Pr(>F)
body_weight	1	16634.20107	16634.20107	38.1430889	0.0001047
I(body_weight^2)	1	2847.58070	2847.58070	6.5296508	0.0286019
Residuals	11	4393.43252	399.40296	NA	NA
Lack of fit	1	32.43252	32.43252	0.0743695	0.7906265
Pure Error	10	4361.00000	436.10000	NA	NA

Now, all but a tiny fraction of the error sum of squares is due to pure error (because the quadratic model captures the shape of the trend better). The lack of fit is nowhere near significant any more. This tells us that with our data, we have little hope of finding a better model than `growth.2` and we should probably end our model-fitting process there.

You might be wondering why we don't follow a procedure like this in general instead of looking at residual plots. The reason is that we are not often in the fortunate position of having replicate observations at the same value of our explanatory variables (or *all* of them if there are more than one). Here we were in that position, and so we could test for lack of fit, but most of the time our explanatory variable values are all, or almost all, different.[9]

It might be worth investigating why there is so much variability in `mcr` among cattle of the same weight — this might be because there is another variable that was not measured, such as (maybe) age: it might be that two cattle of the same weight but different ages would have a different metabolic clearance rate.[10] But that is complete speculation on my part, since we have no data to assess it with.

Tadpoles

A tadpole is a small creature that lives in water, and develops into a frog or toad. Can tadpoles adjust the length of their intestines if they are exposed to a fungus called `Bd`? The dataframe `tadpoles`, shown in Figure 24, contains these variables:

[9] The time when you might have replicated values is in a designed experiment, where you get to control what values you use. For example, you might be testing the effect of temperature in some industrial process, and you might choose to look at six different temperatures, and take several observations at each temperature. That would give you a means of testing for lack of fit. On the other hand, if you have an observational study, you have to accept the values of your explanatory variable(s) as they come, and they are likely to be mostly or entirely different from each other.

[10] The question originally said that the cattle were "of differing ages", but the ages were not recorded, so I took that out to make the question less confusing for you.

- treatment: Bd if the tadpole was exposed to the fungus, Control if not.
- body: body length in mm
- gut_length: length of the intestine in mm (response)
- mouthpart_damage: measure of damage to the mouth (a larger value indicates more damage)

The biological question is whether gut_length has an association with treatment.

(24) (2 points) A regression was fitted to predict gut length from the other variables, with code shown in Figure 25. drop1 output from this regression is shown in Figure 26. What do you conclude? Explain briefly.

Look at the P-values, the same as you would for the summary output (coming later). These are 0.028, 0.046, 0.033, all significant, so all of the three explanatory variables add something to the regression and none of them should be removed. One point for "all significant" and one for something about what that tells us, something like "we should keep them all" or "they all add something to the regression" or "they have a significant relationship with gut_length or similar.

If you prefer, look at the three explanatory variables one at a time, and make the conclusion about each one that it is significant and that it cannot be removed. That's another route to two points.

If you want to only look at the effect of treatment, you need to have a good reason to be only looking at that and not the other two explanatory variables. I could, for example, live with a conclusion like "there is a significant effect of treatment on gut_length after adjusting for body and mouthpart_damage", which implies that the other two explanatory variables are in the regression you are drawing a conclusion from.

(25) (1 point) Why is it better to use the drop1 output to answer the previous question than the summary output shown in Figure 27?

The regression contains a categorical explanatory variable treatment, so we should use drop1 rather than summary. (That's all you need.)

The reason for this is that the summary output only compares the Control group to the baseline Bd group, which might not be enough to assess the overall effect of treatment. In this case, it doesn't actually matter because comparing Bd to Control *is* assessing the effect of treatment: there are only two levels of treatment. So another way to the point is

It doesn't actually matter whether we use drop1 or summary because the categorical variable treatment has only two levels.

The consequence of that is that the P-value for `treatment` is the same, 0.0326, either way. But saying that it doesn't matter because the P-values are the same is only 0.5 because it's `treatment` *having two levels* that makes this happen; you need the insight beyond seeing that the P-values are the same.

(26) (2 points) What would happen if I ran `step` on the model `tadpoles.1`? Explain briefly.

In one word, nothing!

All the explanatory variables in `tadpoles.1` are significant, so `step` would say that nothing further can be removed, and end with the same model as it began.

To demonstrate:

```
step(tadpoles.1, test = "F")
```

```
Start:  AIC=172.23
gut_length ~ body + mouthpart_damage + treatment

                   Df Sum of Sq   RSS    AIC F value  Pr(>F)
<none>                          11829 172.23
- mouthpart_damage  1    2295.3 14124 175.01  4.4630 0.04570 *
- treatment         1    2658.4 14488 175.70  5.1689 0.03265 *
- body              1    2830.2 14659 176.02  5.5029 0.02797 *
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1


Call:
lm(formula = gut_length ~ body + mouthpart_damage + treatment,
    data = tadpoles)

Coefficients:
     (Intercept)              body  mouthpart_damage  treatmentControl
         -20.258             6.442            96.839            25.412
```

Right off the top, the best thing to remove is nothing.

The actual behaviour of `step` is slightly more complicated than this: it actually uses AIC rather than P-values to decide what comes out (one of the columns of output is AIC). AIC is (in this case[11]) dependent on the error sum of squares of the regression and the number of explanatory variables in it. The *smallest* AIC goes with the explanatory variable that should be removed, at the top of the output; in this case, that smallest AIC goes with removing nothing, and if you remove any of the others, you'll make the model fit worse (in AIC terms). The consequence of using AIC rather than P-values is that quite often `step` will end up keeping an explanatory variable whose P-value is around 0.10: not significant by our usual standards, but not *clearly* something that should be removed, and `step` tends to keep things if in doubt. In this case, though, the three explanatory variables are clearly significant, and so there's no way `step` will get rid of any of them.

(27) (2 points) Interpret the number 6.442 in the Estimate column of Figure 27.

This is an ordinary regression slope (since it is for `body` which is quantitative), and thus if body length increases by 1 mm, the intestine length is predicted to increase by 6.44 mm, all else equal.

Notes:

- the best answer decodes the variable names (rather than saying `body` and `gut_length`) so that your reader knows what you are talking about.
- don't forget the "all else equal", or something similar like "if the other variables are unchanged".

Since I said "interpret", meaning "say what it means for the benefit of your reader", you need to get both of those two points in my bullets, and you'll lose a half point for each one of them you don't do.

I realized that my answer will still be correct if you change millimetres to any other unit, so referring to "units" of any kind is also fine, although mm are really the units that make sense in this context.

(28) (2 points) Interpret the number 25.412 in the Estimate column of Figure 27.

This is also a "slope", but of a categorical explanatory variable `treatment`, so this compares predicted gut length for the treatment category shown, Control, with the baseline category `Bd`. That is, all of this, or something equivalent to it:

"tadpoles in the control group are predicted to have an intestine length 25.4mm longer than tadpoles who were exposed to the fungus, all else equal."

Again, "interpret" includes decoding the variable names for your reader. This is work that *you* need to be doing.

[11] You can define AIC for *any* model, like the ones in STAD29, not just for a regression.

One point if you get as far as "predicted `gut_length` is 25.4 bigger for the `Control` group compared to the baseline". 1.5 if you do at least one of these:

- translate the variable names into what they actually are
- say what the baseline actually *is*
- include the "all else equal".

It would have been easier if the actual treatment had been something later in the alphabet than `Control`, but so it is. This was the effect of interest to the biologists, and since the effect of treatment is significant (as we found earlier), it says that tadpoles *can* adjust the length of their intestines if they are exposed to the fungus, by making the length shorter, after adjusting for the effects of body length and mouthpart damage (which might have been different between the two treatment groups, in which case adjusting for them would have been a good idea).

If you are tempted to write "if `treatmentControl` increases by 1", stop and think about whether that makes any sense at all. `treatment` is a categorical variable, so its possible values are `Bd` and `Control`, neither of which can be increased by 1. The only change you can make to a categorical variable's value is to change it from one of its levels (the baseline `Bd` in this case) to another level (`Control` here), and the "slope" says what will happen to the response variable `gut_length` if you do *that*.2

(29) (3 points) A plot is shown in Figure 28. What do you conclude from this plot? Your answer should make it clear that you know what this plot is.

This is a plot of residuals against fitted values (as you see from the code, or from the axis labels on the graph). One point.

What we conclude from it is that there is no pattern (the second point), and therefore that there are no problems indicated (here, at any rate) with the regression, or with the response variable (the third point). For example, you might say that no transformation of the response variable is necessary (since in multiple regression, problems on this plot indicate problems with the response variable).

If you jump straight to the conclusion that there are no problems, you'll lose the point for "no pattern", because it looks as if you are guessing at an answer. At this level, you need to be clear about *how you know* that your answer is correct.

Make it clear that you know this is a plot of residuals *against fitted values*, because the normal quantile plot of residuals and plots of residuals against explanatory variables are also "residual plots". Also, there is nothing in this Figure that says that the residuals have a normal distribution (unless you work very hard to see it); this plot is for seeing whether the residuals have a *random pattern* relative to the fitted values. (The normal quantile plot of residuals, not shown here, will tell you whether or not the residuals have a normal distribution. "Normal" and "random" are describing two different things.)

If you can argue successfully that the points on this plot are *not* random and what that implies, then I am also good with that (even though I disagree with your conclusion in that case).

(30) (3 points) Another plot is shown in Figure 29. What *code* was used to draw this plot?

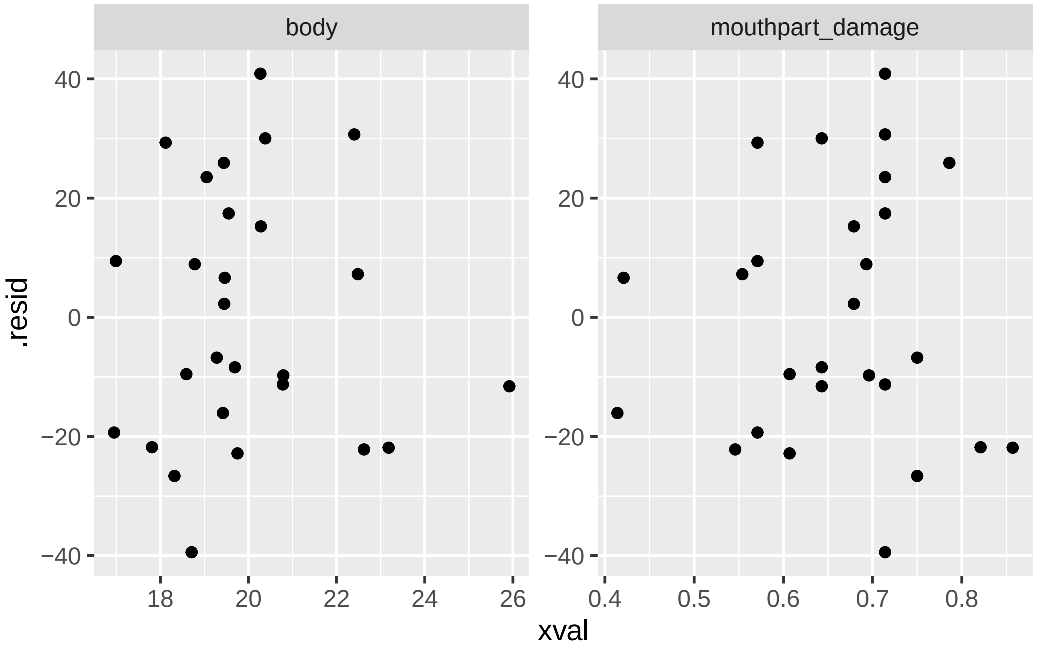
This is a (facetted) plot of residuals against two of the explanatory variables (the two quantitative ones, as it happens). The procedure by which we got these in class was:

- make a dataframe with the original data *and* the residuals in it. I like to use `augment` from the `broom` package for this, but if you have another way that works, I am also good with that. See below for one possibility. (You will note from Figure 1 that `broom` is already loaded, so you are free to use `augment`.) If you skip this step, you have `.resid`, but you don't have the original variables.
- pivot it longer so that all the explanatory-variable values you want are in one column, with the name of the explanatory variable in another column (I called it `xname`). (The clue from the graph is that I used the name `xval` for the column with all the explanatory-variable values in it.) I'm not picky about you using `xname` and `xval` here; use any names you like, but stay with them the rest of the way. You need to explicitly name the two explanatory variables you want, because there is also `gut_length` and `treatment`, and neither of them are in this graph. (You could, I suppose, omit both of those two.)
- plot the residuals against what I called `xval`, facetted by whatever name you gave the column I called `xname`, with `scales = "free"` (which you can tell because each subplot has its own x-scale).

One point for doing each of those; minus a half for each error. (If you make two or more errors on one of these bullet points, you still get 0.5 as long as you have *something* right on that bullet point; on Crowdmark, I used a generic "error" and half-point deduction as soon as I found something wrong on a bullet point, and didn't look further if there was something correct there.)

Thus my code is:

```
tadpoles.1 %>% augment(tadpoles) %>% # first bullet point
  pivot_longer(c(body, mouthpart_damage), # name these two columns and put inside c
               names_to = "xnames", values_to = "xval") %>% # second bullet point
  ggplot(aes(x = xval, y = .resid)) + geom_point() +
    facet_wrap(~ xnames, scales = "free") # third bullet point
```



To satisfy the first bullet point, any way you have of adding the residuals to the original dataframe is good, such as:

```
tadpoles %>% mutate(resid = residuals(tadpoles.1))
```

```
# A tibble: 27 x 5
   treatment  body gut_length mouthpart_damage  resid
   <fct>     <dbl>      <dbl>            <dbl>  <dbl>
 1 Bd         20.3       191.            0.679  15.3 
 2 Bd         19.8       143.            0.607 -22.8 
 3 Bd         19.3       170.            0.75   -6.77
 4 Bd         17.8       152.            0.821 -21.8 
 5 Bd         20.8       171.            0.696  -9.75
 6 Bd         17.0       154.            0.571   9.43
 7 Bd         18.1       181.            0.571  29.3 
 8 Bd         17.0       125.            0.571 -19.3 
 9 Bd         19.4       173.            0.679   2.26
10 Bd         19.4       207.            0.786  25.9 
# i 17 more rows
```

and then use the name *you* gave the residuals the rest of the way. (I don't think *I* showed you this, but one of the TAs might have done, so I am good with it.)

This was, in retrospect, a lot of work for three points.

Extra: I didn't ask for an interpretation of these. My take, for what it's worth, is that these are both random as well, and so there are no problems here either. I didn't ask for an interpretation because I thought too many people would find a problem, like fanning-in on the left one or fanning-out on the right one. I don't think these plots are really evidence of heteroskedasticity because the place where the variability is greatest also has the most observations, and these will tend to look more spread out just because there are more of them (rather than an actual unequal variance).

(31) (1 point) Figure 30 shows another plot from the regression. Why is this one a different type of plot from the ones in Figure 29?

Only one point, so "because the explanatory variable `treatment` is categorical" is all you need. Half a point if you say that the boxplot has a categorical variable without saying what it is, or if you otherwise nearly get to the answer, in my judgement.

Extra: The idea is that you want to plot the residuals against treatment as well (since this is one of the explanatory variables), but because `treatment` is categorical, you cannot use a scatterplot any more, and instead you need a boxplot.[12]

If you try to include `treatment` in the pivot-longer, this happens:

```r
tadpoles.1 %>% augment(tadpoles) %>%
  pivot_longer(c(treatment, body, mouthpart_damage),
               names_to = "xnames", values_to = "xval")
```

```
Error in `pivot_longer()`:
! Can't combine `treatment` <factor<a3ec5>> and `body` <double>.
```

because the values that are going into `xval` are partly categorical (the ones in `treatment`) and partly quantitative (the others), and `pivot_longer` won't let you make a column out of things of two different types. Hence, I did Figure 30 separately. (If I had had two categorical explanatory variables, I would have pivoted them longer too, separately from the quantitative ones, and there would have been no problem in having all text values in the `xval` column.)

(32) (2 points) Does Figure 30 indicate any problem with the regression? Explain briefly.

[12] Or, perhaps, facetted normal quantile plots.

The residuals for both treatment groups should be normally-distributed around zero. The residuals for the Bd group are symmetric about a median close to 0 (good), but the Control group residuals are skewed to the right and have a median less than zero (indicating a problem.)

Hence, pick out the non-normal distribution (one point) and say why it's a problem (the second point). Or, say what you would expect/hope to see (the second point) and how you don't (the first point.)

Exponential growth

Exponential growth is defined by the formula

$$y = y_0(1+r)^t$$

where y_0 is the initial value of some quantity, r is the growth rate per unit time (expressed as a proportion less than 1), t is the amount of time, and y is the final value of the quantity.

(33) (3 points) Write an R function called expgrowth that will accept an initial value, growth rate per unit time, and amount of time (in that order), and will return the final value. Use the same notation for things as in my formula above.

The function itself is a one-liner; the difficulty here is getting the pieces in the right places. Use the same names as given in the formula above, so as not to confuse the grader:

```
expgrowth <- function(y0, r, t) {
  y0 * (1 + r)^t
}
```

Don't forget that you need to explicitly multiply the y0 by the 1 + r, and the 1 + r needs to be in brackets (so that 1 is added to r *before* raising to the power t and multiplying by y0).

I'm looking for:

- the first line (minus a half for a small error like getting the inputs in the wrong order)
- implementing the formula (minus a half per error in this, such as not *explicitly* multiplying (with *) or raising to a power with ^. The latter you have seen in regression, and the former you ought to be able to guess as "the same as in a spreadsheet" even if you don't remember seeing it in this course.)

- returning the result like this as the last line of the function (minus a half if you have saved the result and explicitly `return`ed it: I told you in lecture that this is not R style). Saving the result and "displaying" that result on the last line is fine, like this:

```
expgrowth <- function(y0, r, t) {
  answer <- y0 * (1 + r)^t
  answer
}
```

Standard R practice is to have what you want to return to the outside world, calculated but not saved, on the last line of your function. R actually *does* have a `return()`, but doing something like this, even though it will work (I gave it a different name):

```
expgrowth2 <- function(y0, r, t) {
  answer <- y0 * (1 + r)^t
  return(answer)
}
```

reveals that you have come from Python (where this is how you do it) and have not learned what is clearest in R. 2.5 points if you have done this but your answer is otherwise correct.

Extra: when you are developing functions somewhere other than on exams, it's a good idea to test them on cases where you know what the answer should be. In this case, thinking about *one* time period makes it easy to figure out what will happen. For example, if we start at 60, and we have a growth rate of 50% over that one time period, the growth should be 30 (50% of 60), and the final value should be $60 + 30 = 90$:

```
expgrowth(60, 0.50, 1)
```

```
[1] 90
```

Check.

In a second time period at the same growth rate, the further growth would be 50% of 90, that is 45, so we are now at $90 + 45 = 135$:

```
expgrowth(60, 0.50, 2)
```

```
[1] 135
```

Extra extra: if you have studied finance, the formula for exponential growth:

$$y = y_0(1+r)^t$$

might remind you of the formula for growth of money with compound interest:

$$y = y_0(1+r/n)^{nt}$$

where now y_0 is the principal, r is the interest rate (as a number less than 1), t is the number of years, and n is the number of times interest is compounded per year. (My first formula is really the same as compound interest where interest is compounded *once* per year.) The fact that compound interest is like exponential growth is why financial people like it: your money grows faster and faster. The difference between simple interest and compound interest is this: with simple interest, the interest is only paid on the principal, so if you invest $100 at 5% per year, you earn $5 per year every year. With compound interest, what you make in interest is *added* to the principal, so that in the first year you make $5, which is then added to your $100 to make $105, and in the second year you make 5% *of $105*, which is $5.25. This may not look like much of a difference, but over time it really adds up. (I am assuming interest compounded once per year.)

(34) (2 points) How would you use your function to find the final value after 60 seconds of a quantity that grows at 5% per second, starting from an initial value of 20?

Just use (or "call") your function with the right values in the right order. This is meant to be a gimme:

```
expgrowth(20, 0.05, 60)
```

```
[1] 373.5837
```

You won't know the answer, of course.

You can also name the inputs, so that this is also good:

```
expgrowth(y0 = 20, r = 0.05, t = 60)
```

```
[1] 373.5837
```

and if you do that, the inputs can be in *any* order, such as:

```
expgrowth(r = 0.05, t = 60, y0 = 20)
```

```
[1] 373.5837
```

You should be able to get full marks for this question even if you had no idea how to write your function. You know what the structure of the function should be, what its name is, and (I hope) how to call it, so if you can get the inputs in the right order, or name them, you will be good.

One of these things is two points; a small error, like putting the unnamed inputs in the wrong order, is one. (Using 5 instead of 0.05 is only a half point off.) Strictly speaking, if you write your function in the previous part with the inputs in a different order, and then *use that order here*, you should only get penalized once (you have been consistent with yourself), but you run the risk that the grader will not check back for consistency (they have a lot of papers to get through).

The function you wrote in the previous part will work for any time units (the growth is happening every one time unit, whatever it is).

Extra: even though the quantity is not growing very fast, the power of exponential growth is such that if you watch the quantity long enough, it will eventually become very big. If you remember back to the early days of COVID: because it was so infectious, we were worried that the number of cases bad enough to need hospital treatment was going to get out of hand very quickly, and there was lots of talk about "flattening the curve" (the curve in question being an exponential growth curve) by staying home and isolating as much as possible.

(35) (2 points) Exponential growth only makes sense if the initial value is positive. How would you change your function to give an error if the input value is not positive?

This is what `stopifnot` is for (there are other ways, but this is the one we learned in class). My entire revised function is

```
expgrowth <- function(y0, r, t) {
  stopifnot(y0 > 0)
  y0 * (1 + r)^t
}
```

To verify that it works properly:

```r
expgrowth(20, 0.05, 60)
```

```
[1] 373.5837
```

```r
expgrowth(-20, 0.05, 60)
```

```
Error in expgrowth(-20, 0.05, 60): y0 > 0 is not TRUE
```

When run with the same values as before, the function gives the same answer, but when run with a negative initial value, the function gives an error as it should.

For you, I asked what *change* you needed to make, so I want to see two things:

- the `stopifnot` line (1.5 points if correct)
- where it goes in the revised function (as the first line, or "before the calculation", or something similar that makes it clear that you check `y0` *first*: 0.5 points).

Writing out your entire revised function is also good, and therefore 2 points. (This is the case also if you add the `stopifnot` to whatever you had in the previous part, even if it was wrong.)

Something else that works and is consistent with what we learned in class should also receive credit. We did not use `stop` in class. If you can concoct something with `if` that will work (and is consistent with what we learned in class), that is also good. *Printing* the word "error", and then continuing to run, is only one point, because the whole reason for using `stopifnot` is to make your function *stop* running.

(36) (3 points) What code would use your function, along with some variation on `map`, to create a dataframe that contains a column `time` with values from 0 through 10 (seconds) and the values of a quantity `y` at each of those times that has initial value 20 and grows by exponential growth at a rate of 5% per second?

Create a dataframe with the relevant times in it, and then create a new column that uses the right `map` function to work out the value of `y` at each of those times. I told you what column names to use. Our function returns a decimal number (a `dbl`), so the right function is `map_dbl`:

```r
tibble(time = 0:10) %>%
  mutate(y = map_dbl(time, \(t) expgrowth(20, 0.05, t)))
```

```
# A tibble: 11 x 2
    time     y
   <int> <dbl>
 1     0  20
 2     1  21
 3     2  22.0
 4     3  23.2
 5     4  24.3
 6     5  25.5
 7     6  26.8
 8     7  28.1
 9     8  29.5
10     9  31.0
11    10  32.6
```

The name of the input to your anonymous function (the thing inside `\()`) can be anything, as long as you use the same thing in the right place in your call to `expgrowth`.

Another way to create the dataframe of times is

```
tibble(time = seq(0, 10, 1))
```

```
# A tibble: 11 x 1
    time
   <dbl>
 1     0
 2     1
 3     2
 4     3
 5     4
 6     5
 7     6
 8     7
 9     8
10     9
11    10
```

(that is, 0 to 10 in steps of 1.)

One point for making a dataframe of times, and two for getting the mutate (0.5) and `map_dbl` (1.5) correct. Minus a half point per error, but you should get *something* if one of these things is correct enough even with multiple mistakes.

Extra: because R is "vectorized", sometimes you can do this kind of thing without using `map`. This is actually one of those cases:

```
tibble(time = 0:10) %>%
  mutate(y = expgrowth(20, 0.05, time))
```

```
# A tibble: 11 x 2
    time     y
   <int> <dbl>
 1     0  20  
 2     1  21  
 3     2  22.0
 4     3  23.2
 5     4  24.3
 6     5  25.5
 7     6  26.8
 8     7  28.1
 9     8  29.5
10     9  31.0
11    10  32.6
```

but this doesn't answer the question because I asked you to use some kind of `map`. The point of this question was to see whether you understood how `map` works. Only one point if you do this.

(37) (4 points) Write a function called `plot_expgrowth`, using your previous work, that:

- takes as input an initial value, growth rate, and maximum time
- constructs a dataframe that contains times from 0 up to the input maximum time (in steps of 1 time unit) and values of the quantity at each of those times
- makes a plot of the quantity against time.

This is actually a lot of pretend copying, pasting, and editing of work you've already done. Only the graphing bit is new:

```
plot_expgrowth <- function(y0, r, max_time) {
  tibble(time = 0:max_time) %>%
    mutate(y = map_dbl(time, \(t) expgrowth(y0, r, t))) %>%
    ggplot(aes(x = time, y = y)) + geom_point()
}
```

The first two lines of your function are the ones you used to create your dataframe in question 36 with 10 replaced by `max_time`, 20 replaced by `y0`, and 0.05 replaced by `r`. (This process of replacing values by names is typical in writing a function). The third line can optionally join the points by lines or put a smooth trend through them as well as plotting the points.

Marks:

- 1 for the top line, with inputs clearly labelled
- 2 for adapting your work from the previous part
- 1 for plotting the result suitably (which should be an easy mark to get).

Minus 0.5 per small error, minus 1 if the error is large. I tried to check for consistency with what you did earlier, and if you haven't made any more errors, you shouldn't lose any more points.

Optionally, you can also keep your `stopifnot` line from earlier.

If you are doing this in real life, it is a good idea to have functions that are small and self-contained: that is, they do only one small thing and are as simple as possible. For this kind of problem, you might do this by first making a function that constructs a dataframe: that is, does all the calculation of the exponential growth values:

```
construct_df <- function(y0, r, max_time) {
  tibble(time = 0:max_time) %>%
    mutate(y = map_dbl(time, \(t) expgrowth(y0, r, t)))
}
```

and then making a simpler second function that *calls this one* and then does the plotting:

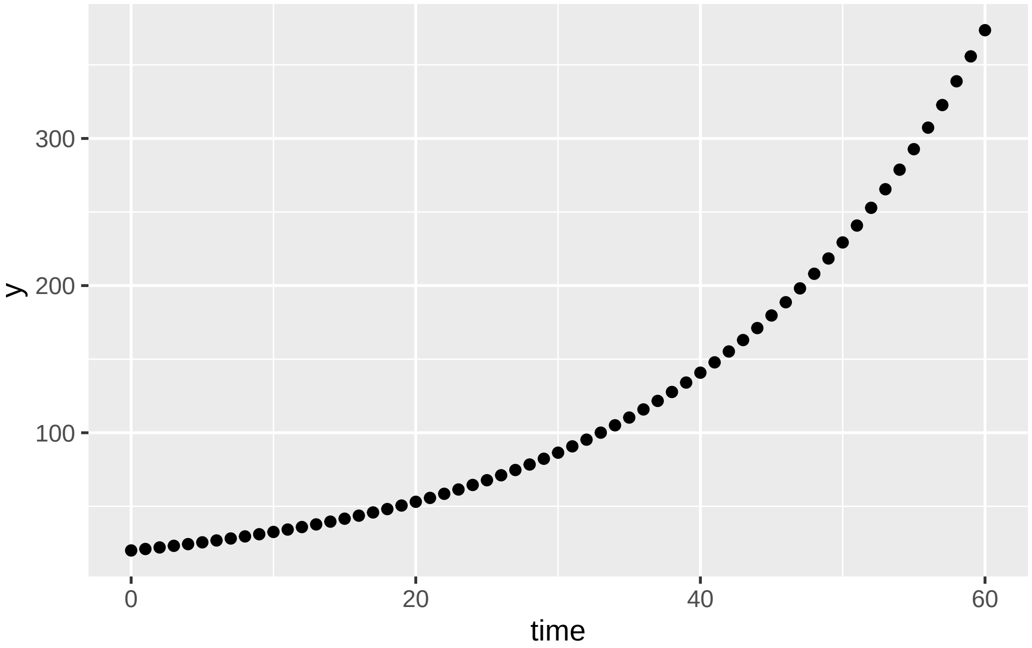
```
plot_expgrowth <- function(y0, r, max_time) {
  construct_df(y0, r, max_time) %>%
    ggplot(aes(x = time, y = y)) + geom_point()
}
```

This is easier *for you* when you come back to it in six months because you can see what it does: it constructs a dataframe (with, presumably, the exponential growth values in it) and then plots it. If there are problems with the construction of the dataframe, you now know to go looking in `construct_df` to see what is going wrong. I doubt you will think to do this on an exam, but full credit if you do and get it right.

If you couldn't do question 36, you can still get full credit here by showing that you know what to do. Show where your code from question 36 would go and what changes you would make to it, and then add the graphing piece.

To verify that my function works (this is actually the second version, but they both work the same):

```
plot_expgrowth(20, 0.05, 60)
```



and you see the exponential growth taking hold.

If you need any more space, use this page, labelling each answer with the question number it belongs to.

Figures

```
library(tidyverse)
library(smmr)
library(broom)
```

Figure 1: Packages

```
species weight
     RT   920
     RT   945
     CH   330
     RT   970
     RT  1240
     RT  1290
     CH   470
     CH   335
     RT  1150
     RT   960
     RT  1130
     RT   925
     RT  1205
     RT  1040
     CH   335
     CH   335
```

Figure 2: Hawks data file (some lines: the remaining lines are in the same format)

```
# A tibble: 3 x 2
  species species_name
  <chr>   <chr>
1 SS      Sharp-shinned
2 CH      Cooper
3 RT      Red-tailed
```

Figure 3: Dataframe `hawk_names`. Note that this has a third species that might have been observed but was not.

```
t.test(weight ~ species, data = hawks)
```

```

    Welch Two Sample t-test

data:  weight by species
t = -32.215, df = 93.635, p-value < 2.2e-16
alternative hypothesis: true difference in means between group CH and group RT is not equal
95 percent confidence interval:
 -715.4837 -632.4050
sample estimates:
mean in group CH mean in group RT
        420.4857        1094.4301
```

Figure 4: Hawks data test

```
# A tibble: 22 x 2
   expend stature
    <dbl> <fct>  
 1   9.21 obese  
 2   7.53 lean   
 3   7.48 lean   
 4   8.08 lean   
 5   8.09 lean   
 6  10.2  lean   
 7   8.4  lean   
 8  10.9  lean   
 9   6.13 lean   
10   7.9  lean   
11  11.5  obese  
12  12.8  obese  
13   7.05 lean   
14  11.8  obese  
15   9.97 obese  
16   7.48 lean   
17   8.79 obese  
18   9.69 obese  
19   9.68 obese  
20   7.58 lean   
21   9.19 obese  
22   8.11 lean   
```

Figure 5: Energy usage data (all)

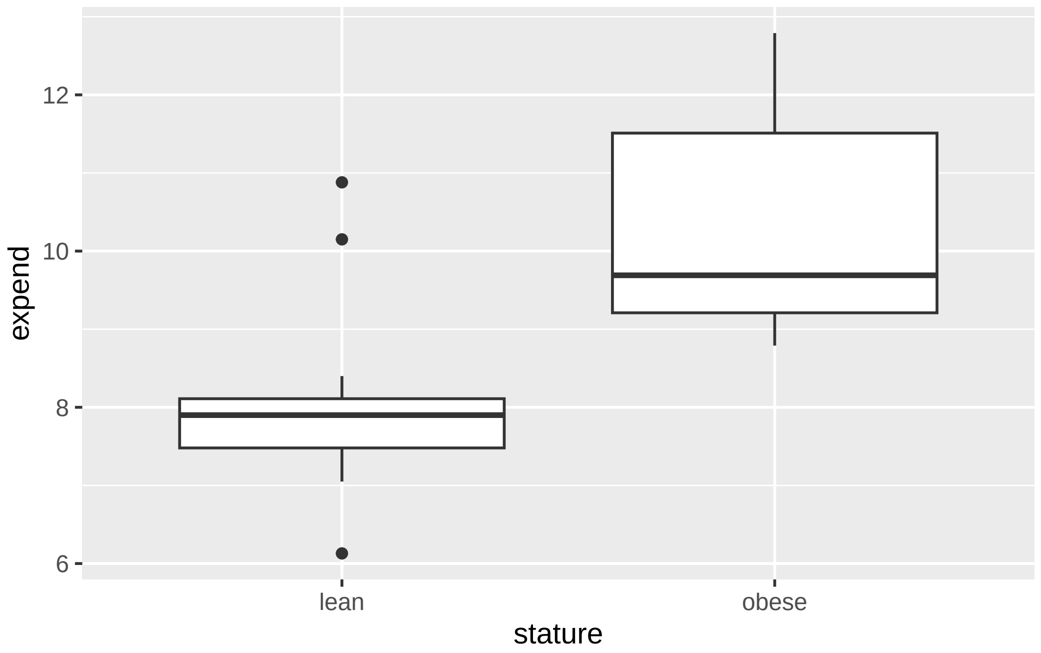


Figure 6: Energy usage graph

```
$grand_median
[1] 8.595

$table
       above
group   above below
  lean      2    11
  obese     9     0

$test
       what        value
1 statistic 1.523077e+01
2        df 1.000000e+00
3   P-value 9.514059e-05
```

Figure 7: Energy usage test

```
# A tibble: 20 x 4
   Subject Educ  Income2005 tr_income
     <int> <chr>      <int>     <dbl>
 1    2188 13-15      36000     13.8
 2    1436 012        10000     10
 3    5307 12         28000     12.9
 4    1222 12         38000     14.0
 5    2944 12         32000     13.4
 6    2809 16+        15000     11.1
 7    1778 13-15       8000      9.46
 8    3036 12         52000     15.1
 9    3023 12         30000     13.2
10    1148 012        31000     13.3
11   12016 16        150000     19.7
12    3949 13-15      39000     14.1
13    3850 12         32000     13.4
14    2057 13-15       9000      9.74
15    3609 13-15      39000     14.1
16    2333 13-15      85000     17.1
17     612 16         46136     14.7
18    3090 12         31600     13.3
19    4822 12         30775     13.2
20    2852 12         33000     13.5
```

Figure 8: Income and education data (20 randomly chosen rows)

```
# A tibble: 5 x 4
  Educ      n income_mean income_sd
  <chr> <int>       <dbl>     <dbl>
1 012     136        12.2      2.66
2 12     1020        13.2      2.55
3 13-15   648        13.8      2.84
4 16      406        15.3      3.30
5 16+     374        15.7      3.35
```

Figure 9: Summary data of transformed income for each education category

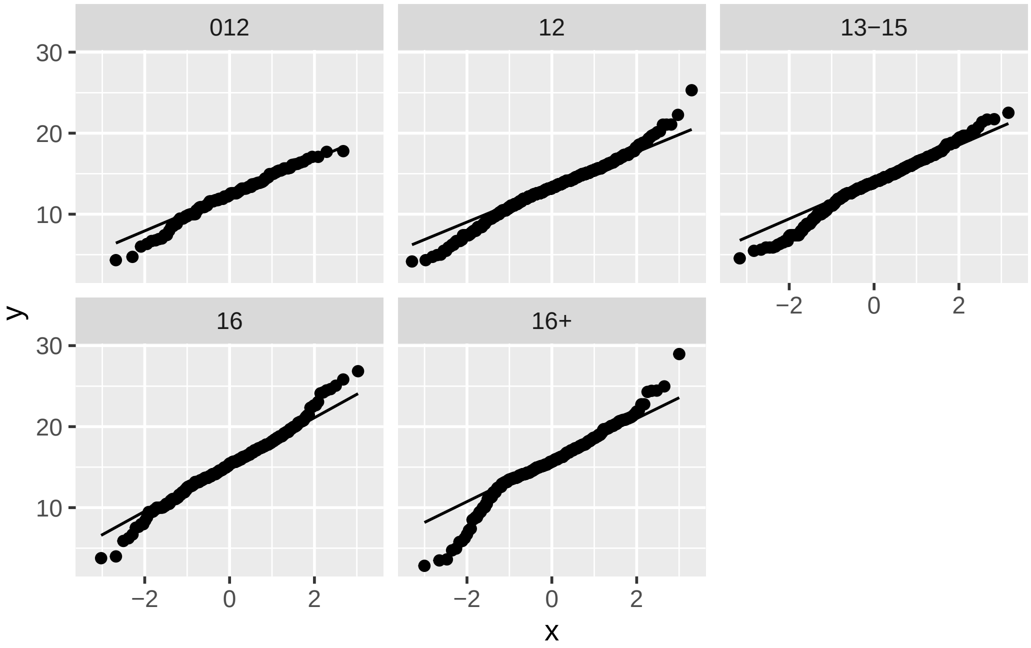


Figure 10: Plots of transformed income

```
              Df Sum Sq Mean Sq F value Pr(>F)    
Educ           4   2902   725.4   87.44 <2e-16 ***
Residuals   2579  21395     8.3                   
---
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 = tr_income ~ Educ, data = incomes)

$Educ
                diff        lwr       upr     p adj
12-012     0.9530098  0.2352841 1.6707356 0.0027287
13-15-012 1.5528225  0.8112558 2.2943892 0.0000001
16-012     3.0464410  2.2674789 3.8254031 0.0000000
16+-012    3.4879841  2.7007045 4.2752637 0.0000000
13-15-12   0.5998126  0.2048468 0.9947785 0.0003375
16-12      2.0934312  1.6320649 2.5547974 0.0000000
16+-12     2.5349742  2.0596996 3.0102489 0.0000000
16-13-15   1.4936185  0.9959743 1.9912627 0.0000000
16+-13-15 1.9351616  1.4245963 2.4457269 0.0000000
16+-16     0.4415431 -0.1219617 1.0050479 0.2039186
```

Figure 11: Income data, analysis 1

```
    One-way analysis of means (not assuming equal variances)

data:  tr_income and Educ
F = 77.507, num df = 4.00, denom df = 677.54, p-value < 2.2e-16


    Pairwise comparisons using Games-Howell test

data: tr_income by factor(Educ)

      012     12      13-15   16
12    0.00111 -       -       -
13-15 4.9e-08 0.00013 -       -
16    7.2e-13 3.9e-10 < 2e-16 -
16+   8.9e-13 3.6e-10 < 2e-16 0.34356


P value adjustment method: none

alternative hypothesis: two.sided
```

Figure 12: Income data, analysis 2

```
# A tibble: 3 x 3
      a     b     r
  <dbl> <dbl> <dbl>
1    12    14     1
2    13    15     2
3    16    17     3
```

```
d1 %>% pivot_longer(a:b, names_to = "trt", values_to = "yield")
```

Figure 13: A dataframe d1 and some code

```
# A tibble: 2 x 4
    x_1   x_2   y_1   y_2
  <dbl> <dbl> <dbl> <dbl>
1    21    22    11    12
2    23    24    13    14
```

Figure 14: Dataframe `d2`

```
# A tibble: 4 x 3
  time      x     y
  <chr> <dbl> <dbl>
1 1        21    11
2 2        22    12
3 1        23    13
4 2        24    14
```

Figure 15: Dataframe `d3`

```
# A tibble: 6 x 3
  rep   g         x
  <chr> <chr> <dbl>
1 r1    trt1     10
2 r2    trt1     11
3 r3    trt1     12
4 r1    trt2     13
5 r2    trt2     14
6 r3    trt2     15
```

Figure 16: Dataframe `d4`

```
# A tibble: 3 x 3
  rep    trt1  trt2
  <chr> <dbl> <dbl>
1 r1       10    13
2 r2       11    14
3 r3       12    15
```

Figure 17: Dataframe `d5`

```
# A tibble: 4 x 3
  u     v         w
  <chr> <chr> <dbl>
1 a     g1       10
2 b     g2       11
3 c     g2       13
4 b     g1       14
```

```
d6 %>% pivot_wider(names_from = v, values_from = w)
```

Figure 18: Dataframe `d6` and some code using `d6`

```
# A tibble: 14 x 2
   body_weight   mcr
         <int> <int>
 1         110   235
 2         110   198
 3         110   173
 4         230   174
 5         230   149
 6         230   124
 7         360   115
 8         360   130
 9         360   102
10         360    95
11         505   122
12         505   112
13         505    98
14         505    96
```

Figure 19: Cattle growth data

```
ggplot(growth, aes(x = body_weight, y = mcr)) + geom_point()
```

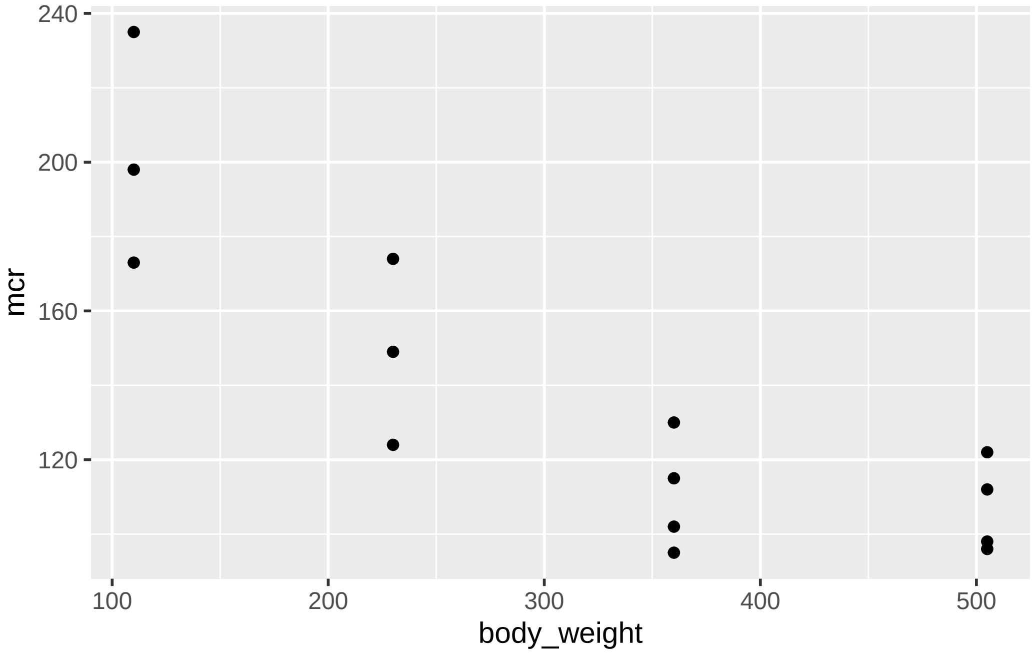


Figure 20: Cattle growth scatterplot

```r
growth.1 <- lm(mcr ~ body_weight, data = growth)
summary(growth.1)
```

```

Call:
lm(formula = mcr ~ body_weight, data = growth)

Residuals:
    Min      1Q  Median      3Q     Max 
-34.553 -13.595   2.138  14.381  48.185 

Coefficients:
             Estimate Std. Error t value Pr(>|t|)    
(Intercept) 212.72093   15.78406   13.48 1.31e-08 ***
body_weight  -0.23551    0.04486   -5.25 0.000204 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 24.56 on 12 degrees of freedom
Multiple R-squared:  0.6967,	Adjusted R-squared:  0.6714 
F-statistic: 27.57 on 1 and 12 DF,  p-value: 0.0002043
```

Figure 21: Cattle growth regression analysis 1

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

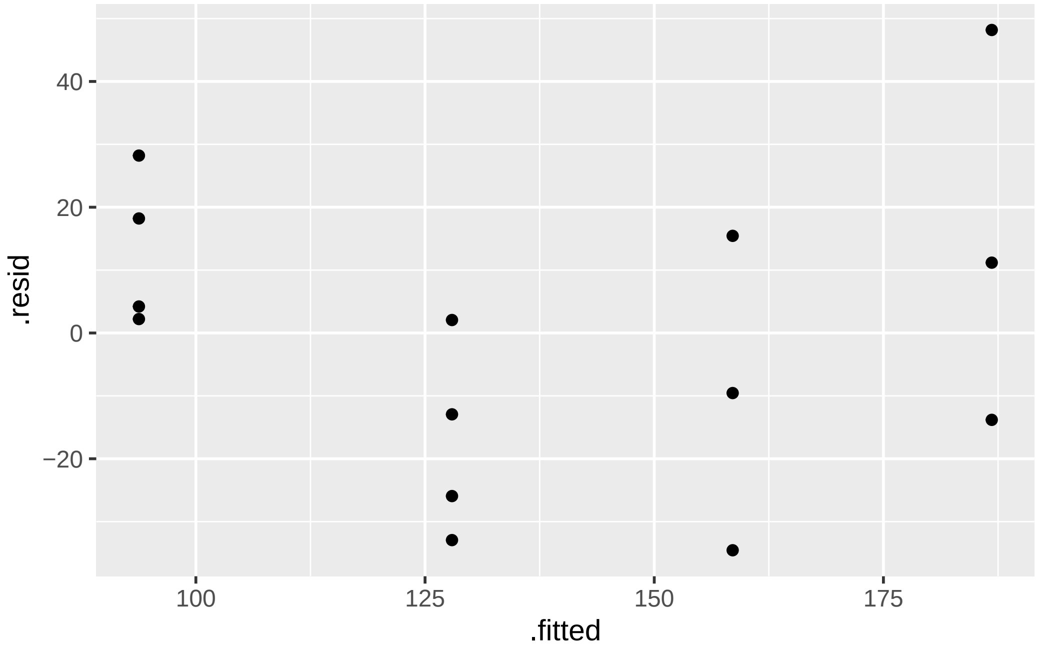


Figure 22: Cattle growth regression plot 1

```r
growth.2 <- lm(mcr ~ body_weight + I(body_weight^2), data = growth)
summary(growth.2)
```

```

Call:
lm(formula = mcr ~ body_weight + I(body_weight^2), data = growth)

Residuals:
   Min     1Q Median     3Q    Max 
-29.89 -10.42  -1.22  13.00  32.12 

Coefficients:
                   Estimate Std. Error t value Pr(>|t|)    
(Intercept)       2.753e+02  2.671e+01  10.305 5.47e-07 ***
body_weight      -7.481e-01  1.954e-01  -3.828   0.0028 ** 
I(body_weight^2)  8.197e-04  3.070e-04   2.670   0.0218 *  
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 19.99 on 11 degrees of freedom
Multiple R-squared:  0.816,	Adjusted R-squared:  0.7825 
F-statistic: 24.39 on 2 and 11 DF,  p-value: 9.051e-05
```

Figure 23: Cattle growth regression analysis

treatment	body	gut_length	mouthpart_damage
Bd	20.28	191.40	0.679
Bd	19.75	142.92	0.607
Bd	19.28	169.81	0.750
Bd	17.81	152.18	0.821
Bd	20.79	171.33	0.696
Bd	16.99	153.92	0.571
Bd	18.12	181.08	0.571
Bd	16.95	124.90	0.571
Bd	19.45	173.06	0.679
Bd	19.44	207.01	0.786
Bd	18.32	143.77	0.750
Bd	20.27	220.35	0.714
Bd	18.71	130.00	0.714
Bd	19.05	195.13	0.714
Control	19.46	177.92	0.421
Control	22.62	181.58	0.546
Control	19.42	154.29	0.414
Control	19.55	217.67	0.714
Control	19.69	185.88	0.643
Control	22.40	249.29	0.714
Control	20.78	196.90	0.714
Control	18.78	202.16	0.693
Control	22.48	210.86	0.554
Control	23.18	215.60	0.857
Control	18.59	174.16	0.607
Control	25.92	222.83	0.643
Control	20.38	228.74	0.643

Figure 24: Tadpole data

```
tadpoles.1 <- lm(gut_length ~ body + mouthpart_damage + treatment,
                 data = tadpoles)
```

Figure 25: Tadpole regression

	Df	Sum of Sq	RSS	AIC	F value	Pr(>F)
	NA	NA	11829.09	172.2270	NA	NA
body	1	2830.183	14659.28	176.0188	5.502892	0.0279750
mouthpart_damage	1	2295.347	14124.44	175.0153	4.462978	0.0457019
treatment	1	2658.389	14487.48	175.7005	5.168863	0.0326488

Figure 26: Tadpole `drop1` output

```
summary(tadpoles.1)
```

```

Call:
lm(formula = gut_length ~ body + mouthpart_damage + treatment,
    data = tadpoles)

Residuals:
    Min      1Q  Median      3Q     Max 
-39.422 -17.701  -6.771  16.338  40.877 

Coefficients:
                 Estimate Std. Error t value Pr(>|t|)  
(Intercept)       -20.258     53.070  -0.382   0.7062  
body                6.442      2.746   2.346   0.0280 *
mouthpart_damage   96.839     45.839   2.113   0.0457 *
treatmentControl   25.412     11.177   2.274   0.0326 *
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 22.68 on 23 degrees of freedom
Multiple R-squared:  0.5502,	Adjusted R-squared:  0.4915 
F-statistic: 9.378 on 3 and 23 DF,  p-value: 0.0003092
```

Figure 27: Tadpole `summary` output

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

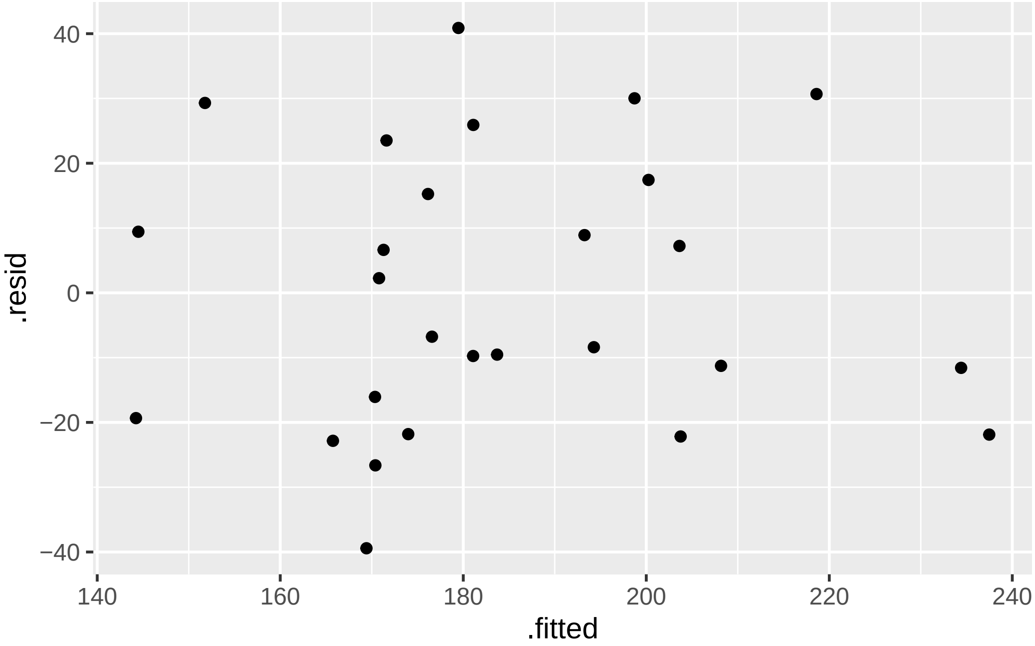


Figure 28: Tadpoles plot 1

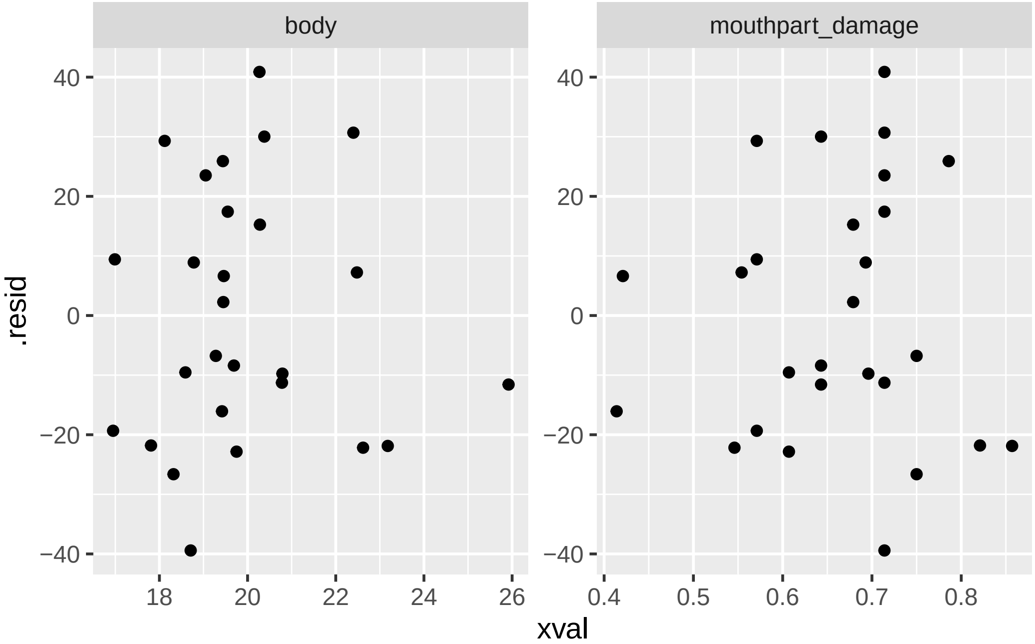


Figure 29: Tadpoles plot 2

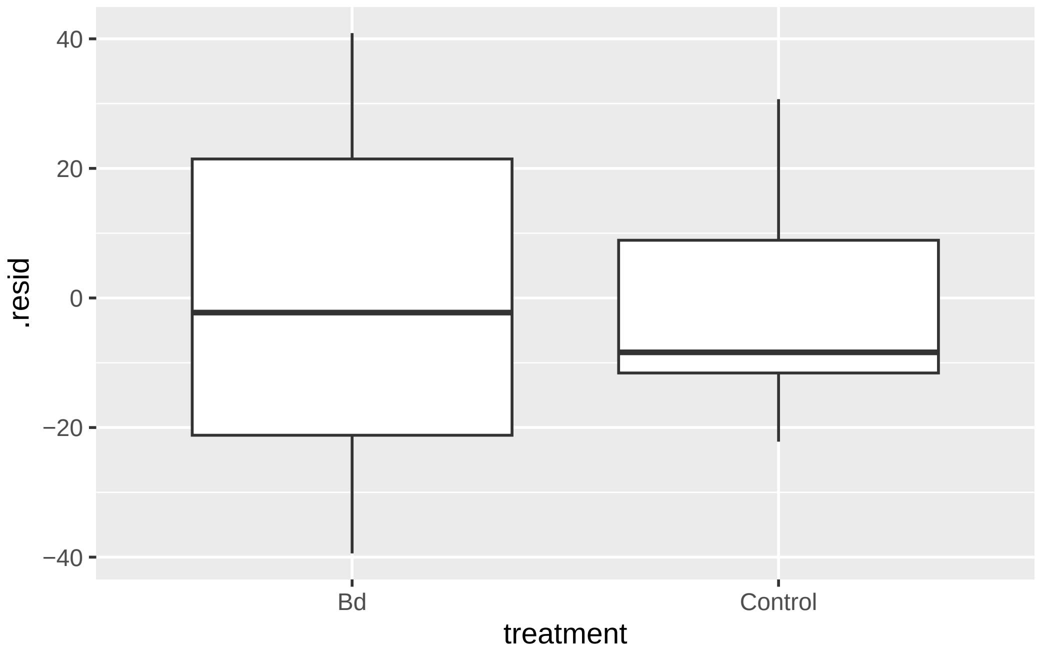


Figure 30: Tadpoles plot 3