University of Toronto Scarborough
Department of Computer and Mathematical Sciences
STAC33 (K. Butler), Final Exam
April 17, 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 xx numbered pages of questions plus this cover page.
In addition, you have an additional booklet of Figures to refer to during the exam.
The maximum marks available for each part of each question are shown next to the question part.
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.

1. Participants in an experiment pretended to be members of a college disciplinary panel judging students accused of cheating. For each suspect, along with a description of the offence, a picture was provided with either a smile or neutral facial expression. Each participant said what they thought was a suitable punishment based on the evidence they had seen and a leniency score was calculated based on the disciplinary decisions made by the participants. (A higher leniency score means a *smaller* punishment.) The facial expression in the picture was also recorded, in `Group`.

   The data file is shown in Figure 2, and is in the file `smiles.txt` in the same folder as your current R Studio project.

   (a) [3] What R code would read the data from the file into a dataframe called `smiles` and display (at least some of) that dataframe?

My answer:

These are aligned columns with variable numbers of spaces in between, and `read_table` is what we used to read in this kind of thing:

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

```
-- Column specification --------------------------------------------------------
cols(
  Group = col_character(),
  Leniency = col_double()
)
```

```r
smiles
```

Group	Leniency
neutral	6.0
smile	3.5
smile	4.5
smile	6.0
smile	4.0
neutral	2.5
smile	7.5
smile	2.5
smile	3.5
neutral	4.0
neutral	2.5
neutral	4.5
smile	3.5
smile	9.0
neutral	3.0
smile	3.0
smile	5.0
neutral	4.5

smile	5.5
smile	5.0

Don't forget to add the name of the dataframe to display it!

Any other way of displaying at least some of the dataframe, such as `glimpse` (that I used with the Blue Jays data in lecture) or `slice`-ing off the first few rows, is acceptable, but just adding the name of the dataframe is definitely enough.

Because the number of spaces between the group name and the leniency score is not constant, any kind of `read_delim` is not going to work. Also, if the data values were *tab*-separated, the `Leniency` numbers would be aligned on the *left* rather than on the right as they actually are.

Two (generous) points for correct code to read the file, and one point for code to display the dataframe in some reasonable way. `read_delim` is only 0.5 out of two even if you get the filename correct.

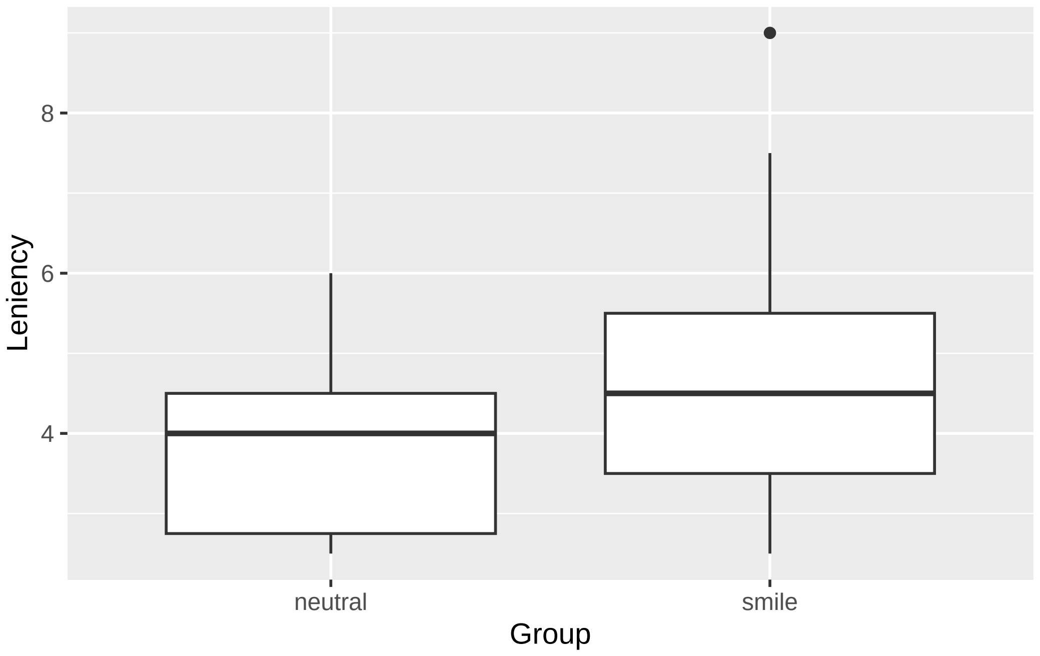
(b) [3] What code will make a suitable graph of the two variables in your dataframe? Justify your choice of graph briefly.

My answer:

One categorical variable `Group` and one quantitative one `Leniency`, so the obvious thing is a boxplot. One point for something like that, including "boxplot" and saying which variable is categorical and which is quantitative (to show that you know). The other two points are for the graph code; see below for the scale.

The plot is this:

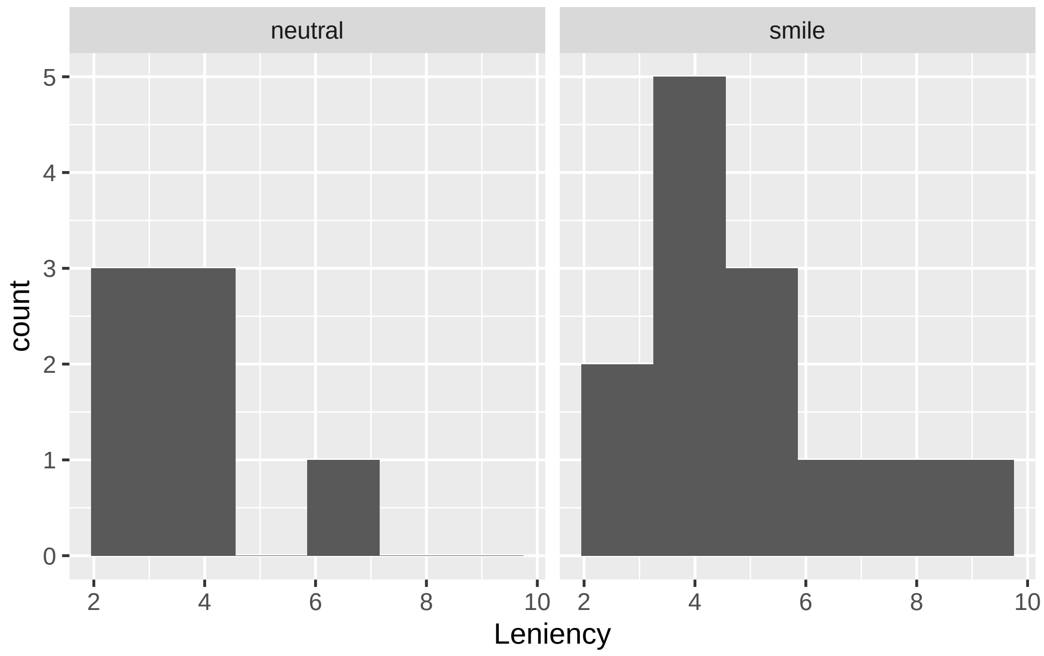
```
ggplot(smiles, aes(x = Group, y = Leniency)) + geom_boxplot()
```



The aim of the study was to compare leniency scores for the two groups of supposed student offenders, and that is what the boxplot does, so I think this is the best graph.

As a (slightly inferior) graph, you might take the attitude that the key variable was `Leniency` and that `Group` is an “extra categorical variable”, so that you could do histograms of `Leniency` facetted by `Group`:

```r
ggplot(smiles, aes(x = Leniency)) + geom_histogram(bins = 6) +
  facet_wrap(~ Group)
```

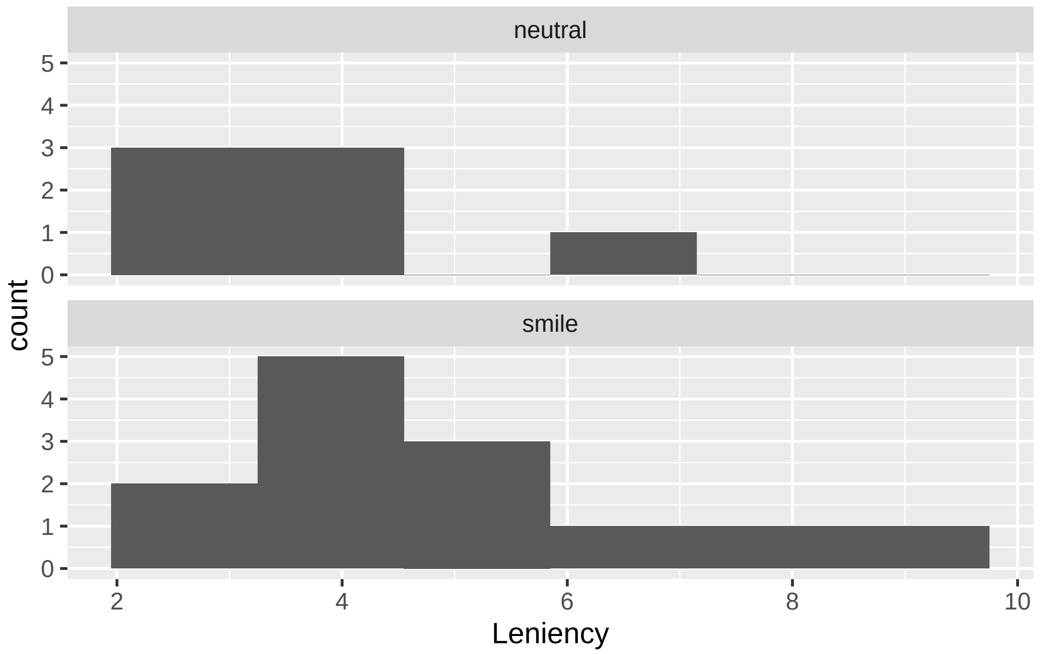


Remember in these ones, you don't mention the variable you're going to facet by until right at the end.

I think it's better to use a few more bins than you normally would, on the basis that the bins are used for *both* histograms and (as it turns out) the `neutral` values don't go up as far and you don't get much of a picture of their shape with, say, 6 bins.

On the basis that we really want to *compare* these two histograms, it is better to put them above and below with a common `Leniency` scale, thus, using one "column of plots":

```
ggplot(smiles, aes(x = Leniency)) + geom_histogram(bins = 6) +
  facet_wrap(~ Group, ncol = 1)
```



Even with this, though, it's not very easy to compare "typical" leniency scores between the two groups. If you were assessing the *shapes* of these two distributions, you could make a case for this being the best graph (boxplots don't show shape very well), but the aim of the study was to see whether punishments are less (leniency scores are higher) on average for the `smile` group, and for that a boxplot is a better graph.

For the graphing part, two points for a boxplot, one point for an above-and-below histogram, 0.5 for a side-by-side histogram, less any deductions for errors.

(c) [3] What code would work out the number of observations in each group, along with the mean leniency score of each group?

My answer:

This is not the kind of case where `count` will work, because you are calculating something else (the mean) along with doing the counting. So you have to use the `n()` idea, like this:

```
smiles %>%
  group_by(Group) %>%
  summarize(n = n(), mean_len = mean(Leniency))
```

Group	n	mean_len
neutral	7	3.857143
smile	13	4.807692

Give the summaries whatever names you like, but `n` is a good name for the number of observations, and something with `mean` in it is a good name for the mean.

One point for the right `group_by`, and two points for a `summarize` with the right two things in it.

Extra: As it turns out, there are only seven observations in the `neutral` group (where the students had a neutral facial expression in their photo), with 13 in the `smile` group. This was not actually all of the original data; I took a random sample from that dataset, so that I could show you all of the data we used here (in Figure 2) to help you decide how you were going to read the values into a dataframe.

(d) [3] What code would display the leniency scores (only) that are (strictly) less than 4 and for observations with a neutral facial expression?

My answer:

This uses `filter` to select the rows you want, with a `select` at the end to display the column you want:

```
smiles %>%
  filter(Leniency < 4, Group == "neutral") %>%
  select(Leniency)
```

Leniency
2.5
2.5
3.0

Have two filters one after the other if you prefer, but the `select` must come at the end, because if it comes before, you won't have a `Group` column to filter on.

I suppose you could also do this:

```
smiles %>%
  filter(Group == "neutral") %>%
  select(Leniency) %>%
  filter(Leniency < 4)
```

Leniency
2.5
2.5
3.0

so that you are only selecting columns after you are done with the ones you are not selecting.

Points: one each for the two filters (together or separately), and one for the select (in a sensible place in your code).

A (hopefully) gentle warmup from the first half of the course.

2. Chromium has been identified as a carcinogen and pollutant. Indoor and outdoor measurements of chromium concentrations (in nanograms per cubic metre) were obtained for a sample of 33 houses in southwestern Ontario, as shown in the dataframe `chromium` in Figure 3. I calculated the column `delta` as the difference, indoor minus outdoor. Each `House` was numbered. The researchers are interested in any differences between indoor and outdoor measurements.

   (a) [2] How do you know that these are matched pairs data? Explain briefly.

My answer:

Each house produced two measurements, an indoor one and an outdoor one. Or, each indoor measurement is paired up with an outdoor one, namely the one that belongs to the same house.

Be specific here about what is paired up with what, and what the individuals are in this case (houses). If your answer is not specific enough, expect one point at best.

   (b) [3] Three normal quantile plots are shown in Figures 4 through 6. Based on the information in the appropriate one(s) of these plots, would you use a matched-pairs t procedure (test or confidence interval), or not? In your answer, state clearly which Figure or Figures you are referring to.

My answer:

A matched-pairs t procedure will be appropriate if the *differences* between the two observations for each individual are close enough to normally distributed, given the sample size. The distributions of the indoor and outdoor measurements individually are *irrelevant*. So we only need to look at Figure 6 and *ignore* the other two Figures.

According to Figure 6, the distribution of differences is either (i) close to normal itself, or (ii) slightly skewed to the left. But we have a sample size of 33 (houses), so we can expect a lot of help from the Central Limit Theorem here. Hence the distribution of differences is close enough to normal, and using a matched-pairs t will have no problems.

So:

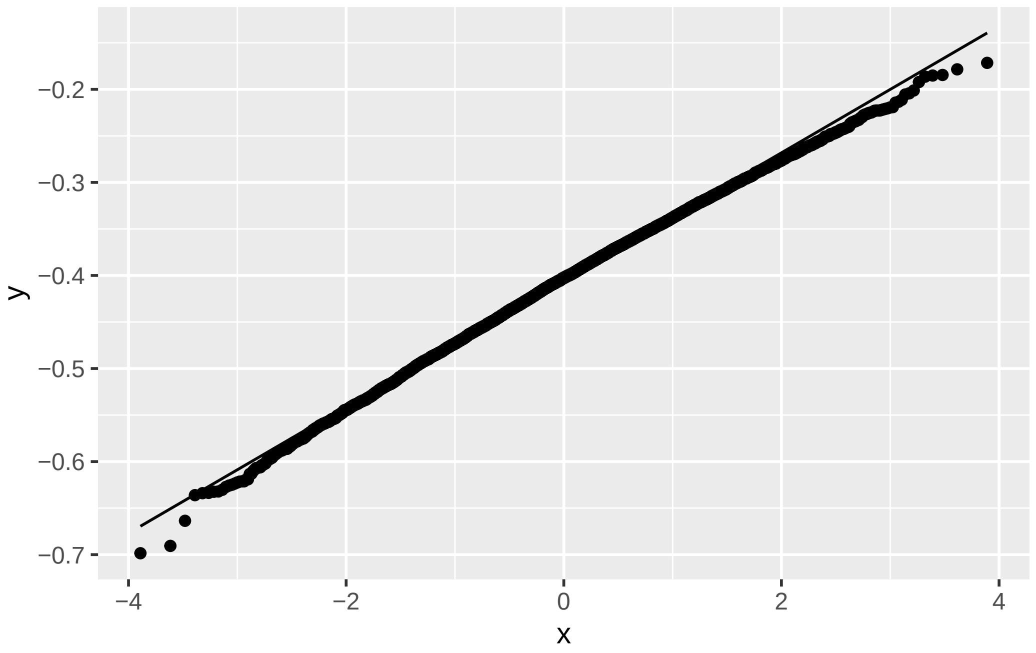
- look at Figure 6 and conclude that it is close to normal or slightly skewed to the left
- don't look at Figures 4 or 5 at all
- note that the sample size of $n = 33$ is going to be large enough to overcome any non-normality you see. Remember there is nothing magic about $n = 30$, and it is an error to pretend that $n = 30$ is special. I suspect that even a sample size of 20 would be large enough in this case, since any non-normality is mild at worst.

One point for mentioning (or appropriately not mentioning) each of those three things. That is to say, if you try to interpret Figures 4 or 5 at all, expect to lose a point.

Extra: if you don't believe me, allow me to show you a bootstrap sampling distribution of the sample mean:

```
Loading required package: lattice
```

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



I did 10,000 simulations because I wanted to use a normal quantile plot to assess normality and I didn't want to get deceived by the tails.

As you see, this is extremely close to normal, providing further confirmation that t is fine here.

(c) [3] What code would run a suitable t-test and obtain a 99% confidence interval for the difference in means between indoor and outdoor measurements? (Do this even if you previously concluded that you should run something other than a t procedure.)

My answer:

The appropriate test is two-sided (see the last sentence of the question), and so you can do the test and confidence interval in one shot (best). There are two choices, the classical paired way:

```r
with(chromium, t.test(Indoor, Outdoor, paired = TRUE, conf.level = 0.99))
```

```
	Paired t-test
```

```
data:  Indoor and Outdoor
t = -5.9509, df = 32, p-value = 1.251e-06
alternative hypothesis: true mean difference is not equal to 0
99 percent confidence interval:
 -0.5929216 -0.2191996
sample estimates:
mean difference
     -0.4060606
```

or as a one-sample *t*-test on the differences, which will mean putting in a null mean difference (or explaining why you don't need one):

```r
with(chromium, t.test(delta, mu = 0, conf.level = 0.99))
```

```

    One Sample t-test

data:  delta
t = -5.9509, df = 32, p-value = 1.251e-06
alternative hypothesis: true mean is not equal to 0
99 percent confidence interval:
 -0.5929216 -0.2191996
sample estimates:
 mean of x
-0.4060606
```

It actually works without a `mu`:

```r
with(chromium, t.test(delta, conf.level = 0.99))
```

```

    One Sample t-test

data:  delta
t = -5.9509, df = 32, p-value = 1.251e-06
alternative hypothesis: true mean is not equal to 0
99 percent confidence interval:
 -0.5929216 -0.2191996
sample estimates:
 mean of x
-0.4060606
```

but if you do it this way, you will need to say that a null mean of zero is the default (otherwise, it looks as if you don't know what the null mean is).

In addition to all of these possibilities, if you don't like `with`, you can use dollar signs throughout (`chromium$delta` and so on), though this starts to get wordy if you do it as a paired test.

In summary, one of these:

- Use `t.test` with the `Indoor` and `Outdoor` columns named and `paired = TRUE`, either using `with` or (two) dollar signs to get the columns from the right dataframe, also obtaining a 99% CI
- Do a one-sample `t.test` on the differences in `delta`, testing a null mean difference of zero (or explaining why you don't need to specify it), either using `with` or a dollar sign to get the column from the right dataframe, also obtaining a 99% CI.

Full credit for realizing that you can do both the test and CI in one shot, 2 points for otherwise correctly doing the test first and then obtaining the confidence interval.

Suggestion for the grader: go through the (virtual) pile of exams once for each solution possibility, so that there isn't too much to keep in your head at once.

(d) [2] My output from the code of the previous part is shown in Figure 7. Interpret the confidence interval in the context of the data.

My answer:

With 99% confidence, the mean indoor concentration of chromium is between 0.219 and 0.593 nanograms per cubic metre lower than the mean outdoor concentration.

(or, the mean outdoor concentration is that much higher than the mean indoor concentration.)

Things I want (expect to lose a half point for any that are missing):

- talk about the indoor and outdoor concentrations specifically, since that is what was measured and was of primary interest, rather than talking about `delta`, which I calculated after the fact from the measurements that were made.
- round the ends of the confidence interval off to a suitable number of decimals. The data are given to 2 decimals, so 2 or 3 decimals is appropriate; more is not.
- in your answer, give the positive values of the numbers in the confidence interval (because "−0.593 to −0.219 higher" is making your reader work too hard)
- give the units of measurement.

For this question, I am happy with "with 99% confidence", since we are talking about one confidence interval obtained from one set of data. If we had been talking about properties of the procedure as a whole, then it would have been right to do the "in 99% of all possible samples" thing.

(e) [1] How could you have guessed that the P-value of the test would be small, if all you could see was the confidence interval?

My answer:

Because the confidence interval does not contain zero. (That's all I need, which is why it was only one point.)

We are highly confident that the indoor measurements are lower than the outdoor ones (or the only plausible values for the mean difference are negative ones), so we would expect the mean difference to be significantly different from zero.

The problem with using a confidence interval to do a hypothesis test is that a confidence interval like this one tells you that the P-value is small (in this case, less than 0.01) but it doesn't tell you *how* small. This might be important, for example if your reader wanted to use $\alpha = 0.001$, and if all you did was give them a 99% CI, they would not be able to tell whether they should be rejecting or not. If you give them the P-value as well, they know what decision to take: even for your reader with their very stringent α, indoor and outdoor are different.

3. 39 quail (small birds) were randomly allocated to one of two treatments (labelled `a` and `b`) for lowering cholesterol. For each quail, the low-density lipoprotein (`ldl`) was measured after treatment. 29 quail were allocated to treatment `a` and 10 to treatment `b`. The researchers are interested in any difference between the two treatments. Some of the data, in dataframe `quail`, are shown in Figure 8.

   (a) [2] A boxplot is shown in Figure 9. Why would you prefer *not* to use a two-sample *t*-test to compare the effects of the two treatments? (I am looking for two distinct points.)

My answer:

- Both distributions have (severe) outliers and are far from normal in shape.
- The sample size for treatment `b` is not large enough to overcome the effect of the outliers on the sampling distribution of the sample mean.

All you need is find *one* of the groups that is not normal enough given its sample size, and the smaller sample size for treatment `b` is the obvious target. Neither distribution is especially normal, but you would need to make the case for treatment `a` being *so* non-normal that even a sample of size 39 was not large enough. (Describing the upper outlier as being "extreme" would do that.)

   (b) [2] Output from Mood's median test is shown in Figure 10. What code produced this output?

My answer:

This:

```
median_test(quail, ldl, treat)
```

You may assume that package `smmr` has been loaded, so that this will work.

Inputs are dataframe, quantitative (response) variable, categorical (explanatory) variable in that order. This is pretty straightforward, so minus one for any mistake(s), as long as you got something substantial correct, otherwise zero.

   (c) [3] Using Figure 10, what do you conclude, in the context of the data?

My answer:

The researchers are interested in any difference between the two treatments, so we use the two-sided P-value of 0.027 as is. This is smaller than 0.05, so there is evidence of a difference in the median `ldl` of the two treatments.

Give the P-value, mention the name of the thing that is different between the two treatments, and say what aspect of it is different between them. One point for each of those. (If this were a one-sided test, you would do the thing about checking that you are on the correct side and then halving the P-value, but it is not, so don't.)

(d) [2] Looking at the rest of the output in Figure 10 (other than the P-value), why is it not surprising that the significance of your test came out as it did? Explain briefly.

My answer:

The majority of the `ldl` values for treatment `a` were above the overall median, and the majority of them for treatment `b` were below, which is unbalanced, so it is not surprising that we inferred a difference between the median `ldl` values of the two groups.

4. When an antibiotic is injected into the bloodstream, a certain part of the antibiotic will bind to serum protein. This binding reduces the medical effect of the antibiotic. The binding rate was measured for 12 cows which were each given one of three types of antibiotics (these originally had long names, but you can use the abbreviated names given in the dataset). A lower binding rate is better (because it means that the antibiotic has an increased medical effect). The data, in dataframe `bind`, are shown in Figure 11.

   (a) [2] A boxplot is shown in Figure 12. The researchers decided that normality was acceptable enough (there were at least no outliers), and so in their opinion it was satisfactory to use some form of ANOVA (in the context of this class, not Mood's median test). Given that, do you see any other relevant issues? Explain briefly. (If you see no issues, explain what issues you were searching for and failed to find.)

My answer:

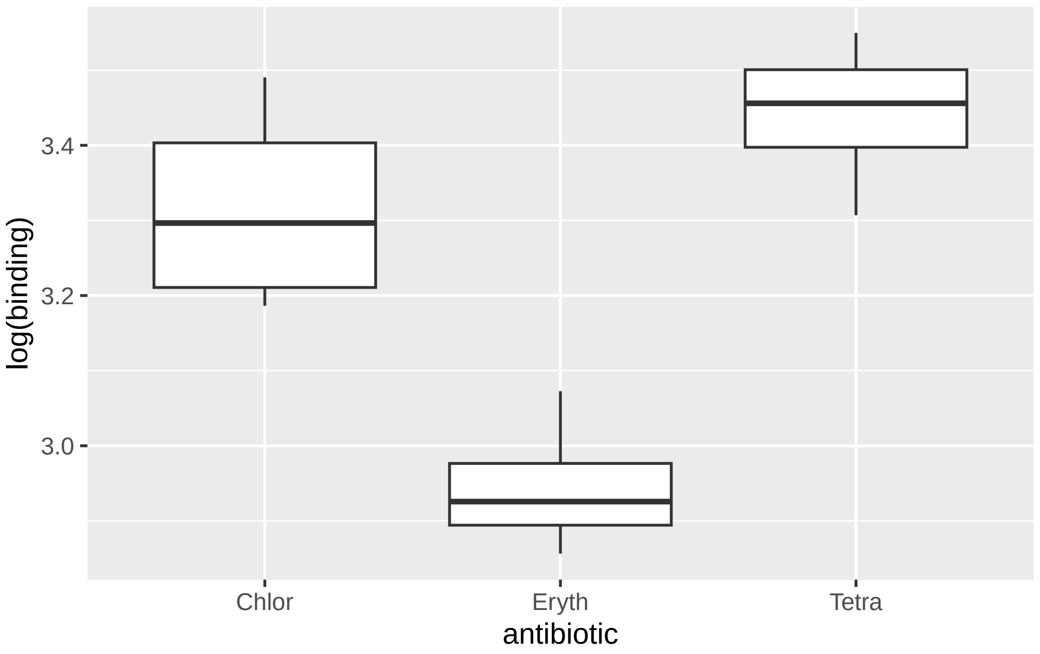
In ANOVA, if normality is all right (which we are saying it is), the next thing to check is equal spreads. The observations for `Eryth` are, according to the boxplot, clearly less spread out than the others (the box is much less tall), and so it is not true that all three groups have equal spread.

You can compare `Chlor` and `Tetra` as well if you like, but this is not necessary: as soon as you have found one group with different spread than the others, the equal spreads assumption fails.

I don't think it's possible to claim here that the spreads are "almost the same", but if that's what you think, make sure to say that you are comparing the spreads.

Extra: the groups with bigger median do at least sort of also have bigger spread, so this is the kind of situation in which a transformation like square root or log might be helpful:

```
ggplot(bind, aes(x = antibiotic, y = log(binding))) + geom_boxplot()
```



This helps a bit, but not enough: now the `Chlor` group is more spread out than the others.

(b) [2] Two possible analyses are shown in Figures 13 and 14. Which of the two analyses is better, and why?

My answer:

The first analysis, in Figure 13, is a regular ANOVA, and the second one, in Figure 14, is a Welch ANOVA. We said (in the previous part) that the group spreads were not all equal, so the Welch ANOVA is better.

Make sure you say:

- which Figure you prefer (by figure number or by analysis number in the caption, 1 or 2)
- what kind of analysis it is
- why what you said before fits that analysis.

The two points are for the second and third bullets.

The fact that the `aov` analysis has smaller P-values is *entirely irrelevant.*

(c) [4] Using your preferred Figure of the previous part and any earlier Figures that are relevant, what do you conclude from the whole analysis, in the context of the data?

My answer:

The conclusions from the Welch test (Figure 14) are:

- The P-value from the F-test is very small (0.0022), so the three antibiotics are not all the same in terms of mean binding
- Therefore we should look at the Games-Howell test to see which ones differ significantly from which
- The antibiotics `Tetra` and `Chlor` are not significantly different, but both the comparisons involving `Eryth` are significant. Give P-values for these if you want, but asserting the significance here is fine.
- Looking back at the boxplot in Figure 12, this is because the antibiotic Eryth has significantly *less* binding than the other two antibiotics.

A point for each bullet.

I don't agree with the `aov` analysis, as discussed before, but if you preferred it previously, you should be consistent and draw your conclusions from Figure 13 instead. Only a few details are different:

- The P-value from the F-test is very small (0.00099), so the three antibiotics are not all the same in terms of mean binding
- Therefore we should look at the Tukey test to see which ones differ significantly from which
- The antibiotics `Tetra` and `Chlor` are not significantly different, but both the comparisons involving `Eryth` are significant. Give P-values for these if you want, but asserting the significance here is fine.
- Looking back at the boxplot in Figure 12, this is because the antibiotic Eryth has significantly *less* binding than the other two antibiotics.

Extra: the `aov` and Tukey analysis has smaller P-values, but the size of the P-values is never a good reason for choosing one analysis over another. The smaller P-values in this case are an illusion because an assumption for the `aov` analysis (equal spreads) is not met.

(d) [2] On the evidence in these data, which antibiotic or antibiotics would you recommend, and why?

My answer:

- The lowest mean binding is best (from the question), and antibiotic `Eryth` has the lowest mean binding (from the boxplot)
- Its mean binding is *significantly* lower than the other two antibiotics (from the Games-Howell, or Tukey if that's what you used).

Picking the lowest one is only one point; you need to say it's *significantly* lower than the others. If it were not, then another experiment might come out with one of the others being best, but because it's significantly lower, we can be confident that this experiment will stand up to replication and that `Eryth` will be consistently the best.

5. In this question, you will (except where stated otherwise) be giving code to convert one dataframe into another. Code that takes the first dataframe and outputs the second one is acceptable. Best answers will accomplish each task with the most direct code possible.

   (a) [3] Dataframe `df1` is shown in Figure 15, and dataframe `df2` is shown in Figure 16. What code would convert `df1` into `df2`?

My answer:

This:

```
df1 %>% pivot_longer(everything(), names_to = "gp", values_to = "y")
```

gp	y
a	10
b	14
c	12
a	7
b	9
c	11

A standard pivot-longer. Any other way that selects all the columns is also good. (Here and elsewhere in this question, saving this in the second dataframe and displaying that dataframe is also good.) Make sure you put the right names in `names_to` and `values_to` to get output columns with the right names.

For this question, expect the grader to be primarily checking that you have the first dataframe, a pipe, the right kind of pivoting, and either displaying directly or saving and then displaying the output.

   (b) [3] Dataframe `df3` is shown in Figure 17, and dataframe `df4` is shown in Figure 18. What code would convert `df3` into `df4`?

My answer:

This:

```
df3 %>% pivot_longer(everything(), names_to = c("gp", "level"),
                     names_sep = "_", values_to = "z")
```

gp	level	z
a	lo	10
a	hi	12
b	lo	7
b	hi	9
a	lo	13

a	hi	14
b	lo	6
b	hi	8

A pivot-longer with two `names_to` columns.

If you don't think of that, there is a less direct route:

```
df3 %>% pivot_longer(everything(), names_to = "var", values_to = "z")
```

var	z
a_lo	10
a_hi	12
b_lo	7
b_hi	9
a_lo	13
a_hi	14
b_lo	6
b_hi	8

Keep in your head (or write down) what this will do, and then think about how to fix it up: the things in the column I called `var` need to be separated-wider:

```
df3 %>% pivot_longer(everything(), names_to = "var", values_to = "z") %>%
  separate_wider_delim(var, names = c("gp", "level"), delim = "_")
```

gp	level	z
a	lo	10
a	hi	12
b	lo	7
b	hi	9
a	lo	13
a	hi	14
b	lo	6
b	hi	8

which gets to the right place. You can see how this is sort of a verbose version of the pivot-longer with two `names_to` columns. Two points at stake if you can do it this way.

(c) [3] Dataframe `df3` is shown in Figure 17, and dataframe `df5` is shown in Figure 19. What code would convert `df3` into `df5`?

My answer:

This:

```
df3 %>% pivot_longer(everything(), names_to = c(".value", "level"),
                     names_sep = "_")
```

level	a	b
lo	10	7
hi	12	9
lo	13	6
hi	14	8

We now want to get columns *called* `a` and `b`, with a column called `level` containing `hi` and `lo`, so we need to use the variant with `.value`.

Pivoting longer and fixing up only sort of works this time:

```
df3 %>% pivot_longer(everything(), names_to = "var", values_to = "z") %>%
  separate_wider_delim(var, names = c("gp", "level"), delim = "_")
```

gp	level	z
a	lo	10
a	hi	12
b	lo	7
b	hi	9
a	lo	13
a	hi	14
b	lo	6
b	hi	8

This is progress, but we have now *gone too far*: we have made it too long. Jot down what this code will produce, and diagnose: we need to pivot `gp` *wider* now:

```
df3 %>% pivot_longer(everything(), names_to = "var", values_to = "z") %>%
  separate_wider_delim(var, names = c("gp", "level"), delim = "_") %>%
  pivot_wider(names_from = gp, values_from = z)
```

```
Warning: Values from `z` are not uniquely identified; output will contain list-cols.
* Use `values_fn = list` to suppress this warning.
* Use `values_fn = {summary_fun}` to summarise duplicates.
* Use the following dplyr code to identify duplicates.
  {data} %>%
  dplyr::group_by(level, gp) %>%
  dplyr::summarise(n = dplyr::n(), .groups = "drop") %>%
  dplyr::filter(n > 1L)
```

level	a	b
lo	10, 13	7, 6
hi	12, 14	9, 8

It is generally a warning sign if you pivot longer and then wider. (There are a few circumstances where you do need to do it, but this is not one of them.) Indeed, this doesn't quite work, because when you do the pivot-wider, there are two values of `z` to go into each cell: the rows are determined by the distinct values of `level`, and there are only two of those.

Two hard-earned points if you get this far, or with some equivalent kind of thinking. This shows, you might say, the value of `.value`!

(d) [3] Dataframe `df6` is shown in Figure 20, and dataframe `df7` is shown in Figure 21. What code would convert `df6` into `df7`?

My answer:

This:

```
df6 %>% pivot_wider(names_from = level, values_from = z)
```

gp	lo	hi
a	10	12
b	7	9

This is a standard pivot-wider, and three rather straightforward points.

(e) [3] A dataframe `df8` and some code is shown in Figure 22. What output will the code produce? (Do not write any code for this question.)

My answer:

This:

```
df8 %>% pivot_wider(names_from = colour, values_from = z)
```

trt	red	green
u	9	13
v	11	17
w	NA	15

The pivot-wider is going to create columns called `red` and `green` (from the contents of `colour`), and will fill them with the values in `z`. The remaining column, `trt`, will determine what row each value of `z` goes in. There are three distinct values in `trt`, namely `u`, `v`, and `w`, so these will be the labels for the rows (and there will be only three rows). My advice is to draw an empty dataframe with columns `trt`, `red`, and `green`, fill in the three distinct values of `trt`, and then, for each value of `z`, put it in the column and row it belongs (the `colour` and `trt` it originally went with). At the end, put an `NA` in any cells that are still empty (one in this case).

The above is the right answer, in the sense that this is what the code actually produces, with the values in `trt` in alphabetical order, and columns named `red` and `green` in that order. If you look at the help for `pivot_wider`, for example here, you see that the column names are by default sorted "by first appearance", meaning that as you read down the `colour` column, the first one you see is `red`, so that the `red` column in the output will be first. However, you can also (once you have learned about it) set `names_sort = TRUE` which will sort the new columns into alphabetical order instead.

As far as I am concerned, it is correct here to get the values of `z` in the right places relative to the value in `trt` and the column name; for example, the value 11 should be in the `v` row and the `red` column, whichever row and column that actually is. But, if you have (especially) the rows in a different order, you run the risk of confusing the grader, who has only a few seconds to look at your answer and decide whether it is correct or not.

6. You might have heard that taller people earn more money. Is it true? The data shown in Figure 23 are a sample of 100 American individuals. For each of these, their annual income (in $), height (inches), weight (pounds), age (years), marital status (categorical), sex (as reported), years of education, and percentile score on the Armed Forces Qualification Test (`afqt`) were recorded. The AFQT is a multiple choice test, administered by the United States Military, used to determine qualification for enlistment in the United States Armed Forces.

   All the individuals reported their sex as either male or female, with, for these data, males being the baseline.

   (a) [2] Some regression output is shown in Figure 24. Is there a significant relationship between income and height, and, if so, is the trend upward or downward? Explain briefly.

   My answer:

   There is a significant relationship, with a small P-value of 0.002 (on the `height` line of the coefficient table).

   The slope, 2484, is positive, so a taller person is predicted to have a higher income: the trend is upward.

   One point each.

   (b) [2] I fitted a second regression by starting with all the explanatory variables and proceeding by backward elimination, reaching the regression shown in Figure 25. How can it be that `height` no longer features in this regression? The best answer will indicate not only what happened, but also why it might have happened, based on what you know or can guess about height.

   My answer:

   This must be because, with `sex`, `education` and `afqt` in the regression, knowing a person's height adds nothing to the prediction. One point.

   In particular, males tend to be taller than females, so height and sex will be associated, and the story we get from here is that males earn more than females (holding education and AFQT score constant), so that the reason taller people earn more is that *males* earn more and males are generally taller than females.

   The second point for saying a decent fraction of this.

   This analysis shows that sex is a stronger predictor than height (because height got eliminated from the regression).

   (c) [3] Interpret the Estimates in Figure 25, in the context of the data. Do not interpret the intercept.

   My answer:

   One point each:

- `sex` is categorical, so females earn less on average than (the baseline) males by about \$23,000, all else equal. (There is unfortunately still sex discrimination in the US.)
- One year more of education is associated with a income increase of about \$4400, all else equal.
- One percentage point (percentile point?) higher on the AFQT is associated with an income increase of about \$330, all else equal.

Words like "all else equal" or "holding other variables constant" should appear in your answer somewhere. Minus a half point if they don't.

(d) [2] Do the signs of the Estimates for `education` and `afqt` (positive or negative) in Figure 25 make practical sense, based on what you know or can guess? Explain briefly.

My answer:

Both Estimates have a positive sign, so:

- having more education should mean having more knowledge, and therefore make an individual more employable (or employable at a better job)
- scoring higher on the AFQT should mean being better suited to life in the military. This might mean that a person is more disciplined or hard-working or better at taking orders (or something along those lines), which would also mean they are better suited to working in the corporate world. Or, a person who scores lower will be less of those things, and though they might be more creative or imaginative, they might have a harder time making a high income. For the second one, use your imagination, ha ha ha.

Something sensible along those lines, or anything else that makes sense to the grader, should get the points here.

(e) [1] Can the regression model in Figure 25 be improved further? Explain (very) briefly.

My answer:

No: all three of the explanatory variables are significant, with P-values less than 0.01.

The intercept cannot be removed, so don't suggest that.

(f) [2] What do you conclude from the plot in Figure 26?

My answer:

This is a plot of residuals against fitted values for the model `heights.1` that we examined earlier. I think you have two choices for the interpretation here, one of these:

- the points form a random pattern and therefore there is no indication of a problem with the regression
- the scatter of points gets slightly wider as we move to the right, so there is a small indication of fanning-out (which might be solved by a transformation of `income`).

If you want to say that this is fanning out, say something about how you know.

(g) [2] What code would produce the plot in Figure 27, bearing in mind that it came from the regression model shown in Figure 25? (Hint: why would you draw a plot of this type in this context?)

My answer:

This is a normal quantile plot, and the standard reason for drawing one in this context is to check the normality of the *residuals*. Hence:

```r
ggplot(heights.1, aes(sample = .resid)) + stat_qq() + stat_qq_line()
```

(h) [2] A third plot is shown in Figure 28. What do you learn from this plot? Explain briefly.

My answer:

This is a plot of the Box-Cox transformation. The peak is near 0.5 (0.5 is in the confidence interval), saying that the regression could be improved by using the square root of income instead of income itself.

There are some clues that this sort of thing might be useful:

- the slight fanning-out on the residual plot
- the slight right-skewness on the normal quantile plot of the residuals
- outside of the analysis here, noting that incomes often have a right-skewed distribution (they can be very big) and often benefit from some kind of transformation.

7. What integers less than 100 are prime (that is, are (evenly) divisible only by 1 and themselves)? We will be using R to determine this, and to do so we will need to write a function to determine whether a particular integer `n` is prime.

   (a) [4] Write an R function called `is_prime` that accepts as input a number `n` and returns TRUE if the number is prime and FALSE if it is not prime. To do that, make a dataframe with all the integers `r` from 2 up to and including the square root of `n`, work out the remainder when `n` is divided by each value `r`, and use the results of the remainder column to decide whether `n` is prime or not. Hints: (i) R has functions `any` and `all` that accept a vector or column. `any` returns TRUE if any of its inputs are TRUE, and `all` returns TRUE if *all* of its inputs are TRUE, (ii) in `:`, real numbers are rounded down, so that for example `2:sqrt(15)` returns `2 3`.

My answer:

My function is this:

```
is_prime <- function(n) {
  tibble(r = 2:sqrt(n)) %>%
    mutate(rem = n %% r) %>%
    mutate(divides = (rem == 0)) -> d
  !any(d$divides)
}
```

My steps:

- use the second hint to make a dataframe containing the values of `r` I will want to divide by
- work out the remainder when the input `n` is divided by each `r`. We used the `%%` operator in the Collatz conjecture example, to work out whether an integer was odd or even.
- create a column `divides` that is TRUE if the remainder is zero (FALSE otherwise)
- if any of the values in `divides` are TRUE, return FALSE (`n` is not prime), but if they are all false, return TRUE. (This is a bit complicated; you might instead define something like "does not divide", and then return TRUE if all of those are true.)

Remember also to make sure that your function returns a single TRUE or FALSE, not a dataframe containing a single true or false. `all` and `any` work on a single column, so it seemed best to me to save the dataframe, and then run `any` on the column I wanted from that saved dataframe. If you have a different way and it will work, you are good. (You may be able to use `pull` or something similar.) You may be able to shortcut defining an extra column as I did when defining `divides`, for example by using `any(rem == 0)` or `all(rem != 0)`.

Points: 2 for getting as far as working out a column of remainders, the other 2 for doing something useful with them and returning something that will be TRUE if the input is prime and FALSE otherwise.

(b) [1] How would you use your function to determine whether 21 is prime?

My answer:

Call your function with 21 as input:

```
is_prime(21)
```

```
[1] FALSE
```

Of course, 21 is not prime because it is divisible by 3. (You don't need to get the output, because you won't see than on an exam.)

This you should be able to get even if you have no idea how to do the previous part, so it is almost a free point.

(c) [4] Use your function `is_prime` to obtain a vector of all the prime numbers less than or equal to 100, using a `map`. `is_prime` returns a TRUE or FALSE value, which in `map` terms is called a `lgl` ("logical").

My answer:

The first way I thought of is via making a dataframe:

```
tibble(x = 1:100) %>%
  mutate(prime = map_lgl(x, \(x) is_prime(x))) %>%
  filter(prime) %>%
  pull(x)
```

```
 [1]  3  5  7 11 13 17 19 23 29 31 37 41 43 47 53 59 61 67 71 73 79 83 89 97
```

The steps here are:

- create a dataframe with all the numbers you want to test for primeness
- create a column that says whether or not each one is prime (this is where `map` gets used; the appropriate one is `map_lgl` because `is_prime` returns a `lgl`)
- grab only the ones that actually are prime
- pull out just the values of `x` (we know that `prime` is TRUE for these).

You can also do it without a dataframe, but there is a step involved that you may not know how to do:

```
x <- 1:100
x %>%
  map_lgl(\(x) is_prime(x)) %>%
  x[.]
```

```
 [1]  3  5  7 11 13 17 19 23 29 31 37 41 43 47 53 59 61 67 71 73 79 83 89 97
```

The steps here are:

- set up a vector of numbers you want to test for primeness (this needs to have a name because we refer back to it later)
- "for each thing in `x`, work out whether it is prime or not" (this produces a vector of TRUE and FALSE as long as `x` was)
- return the values of `x` that correspond to the TRUE things.

I haven't shown you how to do the last step in this course, but if you remember it from somewhere else, I have no problem if you use it here. The `.` means "what came out of the previous step" (which I think you *have* seen in this course), and putting it inside `x` as subscripts means to return the values of `x` that have a TRUE subscript (this part you may have seen somewhere else, if not in this course).

Two points for running `map_lgl` appropriately to get an indication of whether each number from 1 through 100 is prime, and the other two points for turning *that* into a list of the actual prime numbers. (If you go the dataframe way, one out of the last two for doing something like a `filter` to obtain only the rows corresponding to the prime numbers.)

8. A process produces observations that are either successes or failures, independently with probability p of being a success. The total number of successes x in a total of n observations is recorded. Our aim is to estimate p using Bayesian methods with Stan.

   In Stan, the binomial distribution is called `binomial`, with two inputs that are the number of trials and the success probability in that order. Also, the uniform distribution is called `uniform`, with two inputs that are the lower and upper limits of the uniform distribution.

   (a) [4] Write a complete Stan program to obtain the posterior distribution of p, given inputs of n and x. Before collecting the data, you think that p could take any of its possible values, so give it a suitable uniform distribution. Make sure to supply lower and/or upper limits on variables that should have them.

My answer:

The description of the data-generation process means that x has a binomial distribution with parameters n and p, which you use for the likelihood. The business about "before collecting the data" means that you use a uniform distribution for the prior; since p is a probability, the limits of the uniform distribution should be 0 and 1.

Furthermore, the only parameter is p, which is a real number between 0 and 1. n and x are both data and are integers. So my Stan program looks like this:

```
data {
  int<lower = 0> n;
  int<lower = 0, upper = n> x;
}


parameters {
  real<lower = 0, upper = 1> p;
}

model {
  // prior
  p ~ uniform(0, 1);
  // likelihood
  x ~ binomial(n, p);
}
```

p needs to have lower and upper limits (in the `parameters` section), but I don't need to see the lower and upper limits in the `data` section. (In practice, all limits on variables in the data section do is to get Stan to check your input data and see that it satisfies the limits, giving an error if for example x is bigger than n. The important limits are the ones on p, because you don't want the sampler to stray into negative values, or values bigger than 1, for p.)

The grader will award you a mark according to how much of this you have correct (in their estimation). I think it is wrong to set `x` up as an array, unless you are *very* careful about what your data look like (see comments below the next part). The data we have are a single total number of successes and a single total number of trials, so your code should reflect that.

If you wish to avoid hard-coding the 0 and 1 in the uniform distribution, call these values (say) `a` and `b`, set the prior for `p` to be `uniform(a, b)`, and input the values 0 and 1 for `a` and `b` in your data list (next part).

I ought to have told you what name to give this, but if you seem to have given it a sensible one (in (c)), I am happy with that.

(b) [2] You observed 30 successes in 50 trials. How would you set up this data as an object `my_data` to input into Stan?

My answer:

As a `list`:

```
my_data <- list(x = 30, n = 50)
my_data
```

```
$x
[1] 30

$n
[1] 50
```

The `n` and the `x` can be either way around, but they both need to be there. You don't need to display the result.

I made this one easy for you, because there are no arrays anywhere in sight. If you think there should be, you need to say something about what's in them: for example, if your `x` is an array, you need it to be filled with 30 1 values (for successes) and 20 0 values (for failures), consistently with your code in (a). If your (a) and (b) are consistent in this regard, you are good (I checked; I think about one student made it work this way). In your previous work with the binomial distribution, if you *counted* the number of successes, your data would be two scalars (the total number of successes and the number of trials), but if you listed the result of each trial as a success or failure, your data would be a scalar (`n`) and a vector (array) of length n whose *i*-th element is the result of that trial. The two ways are actually equivalent, because the total number of successes is the sufficient statistic for estimating `p`, and you don't need the results of the individual trials. (The concept of "sufficient statistic" is actually not a frequentist thing but a likelihood thing: two datasets with the same sufficient statistic have the same likelihood, and so if you don't change the prior they produce the same posterior distribution as well.)

Your (a) needs to include the `data` section. If you put it here, you'll get credit for (a) for it (if it is correct), but you won't get any credit for (b).

The things contained in your `list` here should be there if and only if they are contained in your `data` section in (a), and they must be of the same type (eg. array, if your data section contained an array). For example, if you used a more general `uniform(a, b)` prior distribution in (a), `a` and `b` need to be part of your `list` here.

(c) [2] What R code would compile your Stan program and sample from the posterior distribution (using the default values)? Call your compiled Stan program `binom`.

My answer:

Compilation is this, for one point:

```
binom <- cmdstan_model("binom.stan")
```

and the sampling is this, for the second point (saving the output is optional for this question, though in practice you would save it):

```
ans <- binom$sample(data = my_data)
```

For the `data`, I asked you to call it `my_data`, so use that name here as well. If you had a compilation line with a sensible looking name for your Stan file, and a sampling line with a `data =` and the name of the data you made in the previous part, then you were good for two points, though having said that, a lot of people forgot to do one of the two things.

Note that you can get two points here even if you got *neither* of (a) and (b). I am not checking your Stan code or your data for correctness here, since I already did that; if you have the right structure, you have two points regardless of what happened before.

This one was fast to mark, since most people got 1 or 2 from doing one or two of the things. If you made a mistake in either of the bits of code, you might have lost another half point. (In contrast to marking (a), I have to say, where there were a thousand and one ways to attempt it, and I had to find some points for each of them no matter how off-base.)

(d) [2] The output from your Stan model is shown in Figure 29. How would you interpret the numbers in the `q5` and `q95` columns of the `p` row?

My answer:

From a Bayesian point of view, `p` is a random variable, so we can make probability statements about it. `q5` and `q95` are the 5th and 95th percentiles of the posterior distribution, so the best answer is: there is a 0.90 probability that p lies between 0.48 and 0.70.

Note that you don't need to hedge this about with "in 90% of all possible samples" as you do with a confidence interval, and indeed the word "confidence" should not appear in your answer at all. This may involve mentally un-learning some of the things you learned earlier! (The randomness here is in a different place than in "frequentist" or repeated-sampling statistics.)

One point for something like "the posterior interval for p is from 0.48 to 0.70. This is true, but not very insightful.

Extra: the posterior mean is close to 0.60, which is the maximum likelihood estimate ($30/50 = 0.60$). What has happened here is that we had a vague prior distribution ("p could be anything between 0 and 1 with equal probability") and we had a decent amount of data (50 trials), so that the inference is dominated by the data rather than the prior. When you have a lot of data, it doesn't really matter what the prior distribution is. The posterior distribution is really a weighted average of the prior and the likelihood (in some loose sense), so that when you have a vague prior and lots of data, the posterior is a lot like the likelihood. On the other hand, if you were fairly sure that p was near, say, 0.5, before you looked at any data, and if you didn't have much data, the posterior mean would also be close to 0.5 pretty much whatever the data say. Thus, it is important to have a prior that really does reflect your prior beliefs (ignorance about p in this case).

Bayesian statistics has a whole field called "prior elicitation", in which the statistician has ways to get at what the user really thinks the parameters might be, prior to looking at any data. In this case, the user might say "I'm pretty sure that p is between 0.3 and 0.7" (based on their knowledge of the process producing the data, whatever it is), and the statistician might take that as the middle 95% of the user's prior distribution, and take a prior distribution that is normal with that middle 95% (mean 0.5 and SD 0.1, roughly). In that case, the posterior mean would be somewhere between 0.5 (prior mean) and 0.6 (maximum likelihood estimate), closer to the latter because there is a fair amount of data.

Use this page if you need more space. Be sure to label any answers here with the question and part they belong to.

Numbered Figures begin here, in with caption and label:

```r
library(tidyverse)
library(smmr)
library(PMCMRplus)
library(MASS)
library(cmdstanr)
```

Figure 1: Packages

```
Group   Leniency
neutral        6
smile        3.5
smile        4.5
smile          6
smile          4
neutral      2.5
smile        7.5
smile        2.5
smile        3.5
neutral        4
neutral      2.5
neutral      4.5
smile        3.5
smile          9
neutral        3
smile          3
smile          5
neutral      4.5
smile        5.5
smile          5
```

Figure 2: Smiles leniency data

House	Indoor	Outdoor	delta
14	0.18	0.66	-0.48
26	0.28	0.48	-0.20
21	0.23	0.12	0.11
4	0.12	0.54	-0.42
15	0.18	0.29	-0.11
33	0.62	0.36	0.26
22	0.23	0.54	-0.31
11	0.17	0.32	-0.15
13	0.18	1.55	-1.37
3	0.09	0.47	-0.38
17	0.19	1.02	-0.83
19	0.22	0.90	-0.68
27	0.29	0.27	0.02
28	0.34	0.37	-0.03
9	0.15	0.86	-0.71

Figure 3: Chromium data (randomly chosen rows out of 33 total)

```
ggplot(chromium, aes(sample = Indoor)) + stat_qq() + stat_qq_line()
```

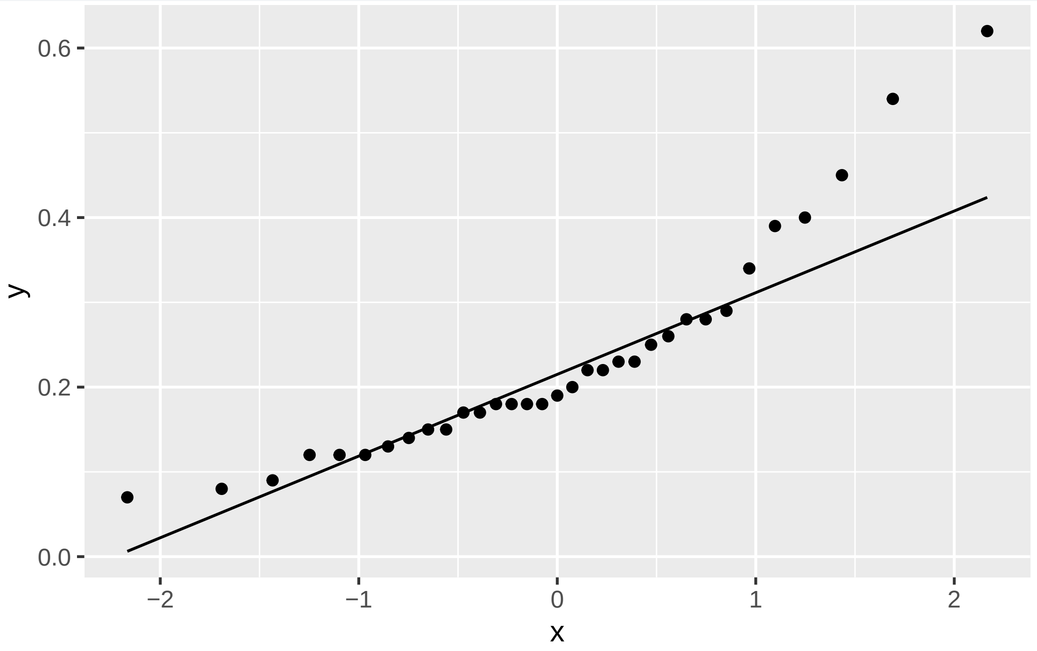


Figure 4: Chromium data normal quantile plot 1

```
ggplot(chromium, aes(sample = Outdoor)) + stat_qq() + stat_qq_line()
```

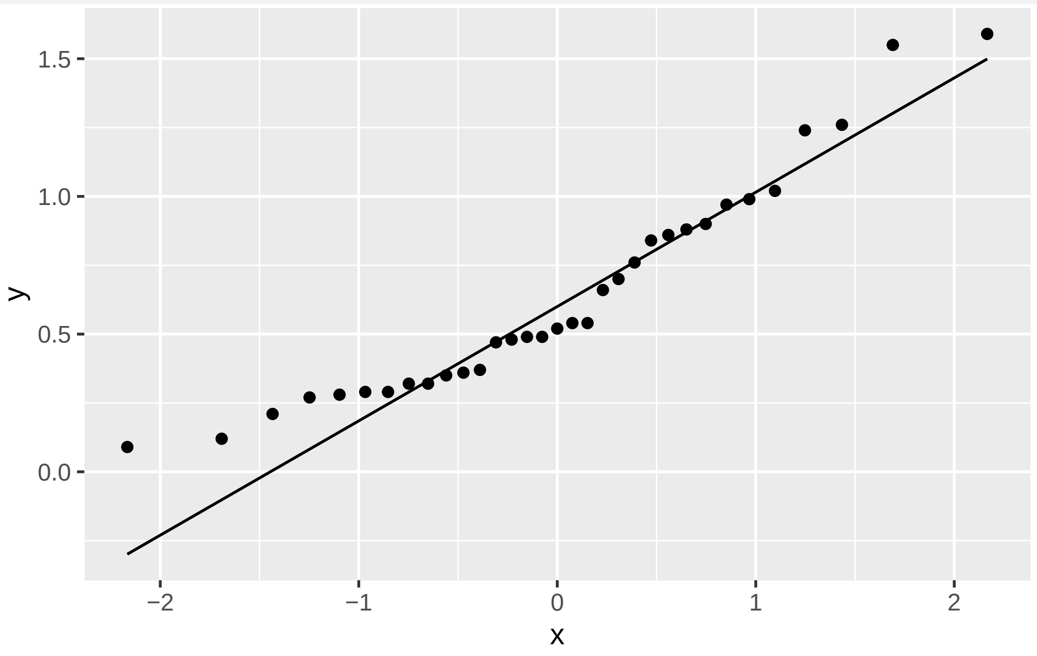


Figure 5: Chromium data normal quantile plot 2

```
ggplot(chromium, aes(sample = delta)) + stat_qq() + stat_qq_line()
```

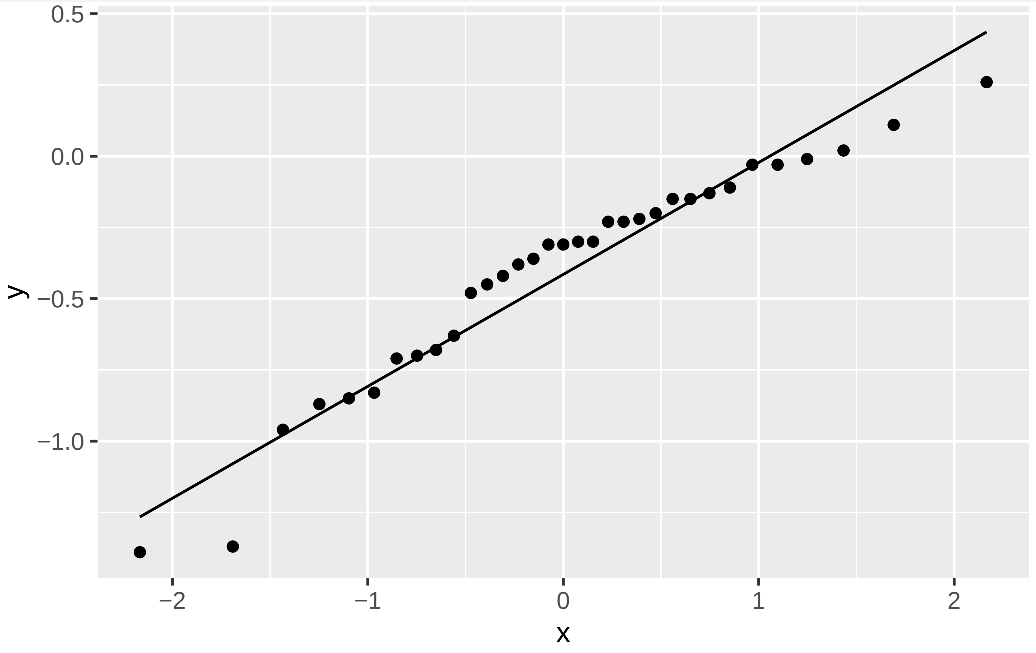


Figure 6: Chromium data normal quantile plot 3

```
    Paired t-test

data:  Indoor and Outdoor
t = -5.9509, df = 32, p-value = 1.251e-06
alternative hypothesis: true mean difference is not equal to 0
99 percent confidence interval:
 -0.5929216 -0.2191996
sample estimates:
mean difference
     -0.4060606
```

Figure 7: Chromium data t output

treat	ldl
a	66
b	30
b	36
a	50
a	52
a	73
a	58
b	33
b	98
a	54

Figure 8: Quail data (some)

```r
ggplot(quail, aes(x = treat, y = ldl)) + geom_boxplot()
```

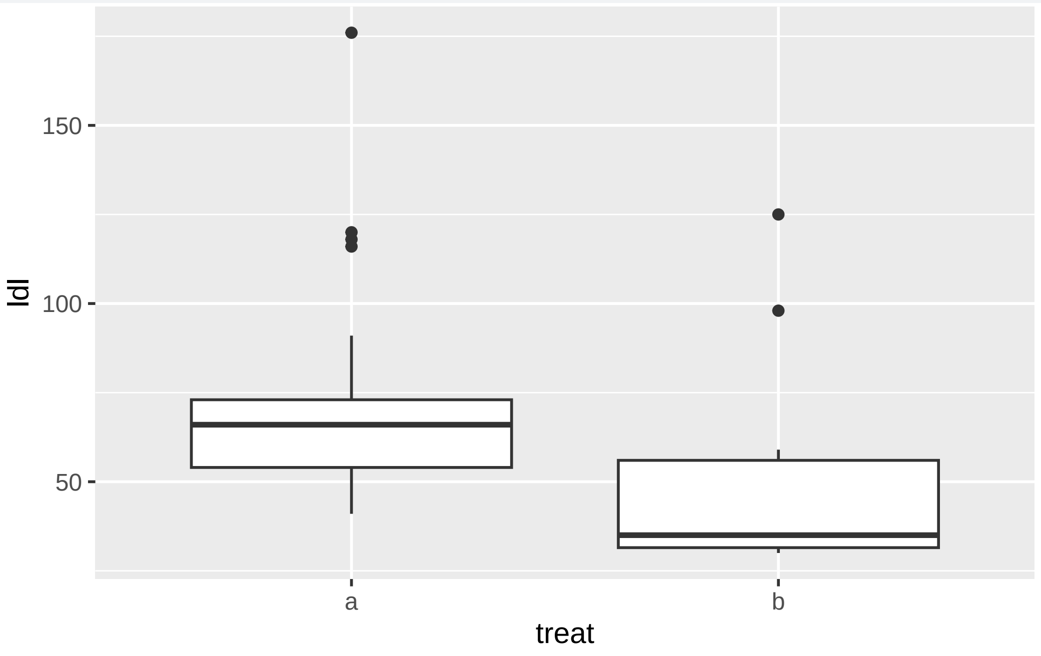


Figure 9: Quail data boxplot

```
$grand_median
[1] 62

$table
     above
group above below
    a    17    11
    b     2     8

$test
       what      value
1 statistic 4.88571429
2        df 1.00000000
3   P-value 0.02707983
```

Figure 10: Mood's median test for quail data

antibiotic	binding
Tetra	27.3
Tetra	32.6
Tetra	30.8
Tetra	34.8
Eryth	21.6
Eryth	17.4
Eryth	18.3
Eryth	19.0
Chlor	29.2
Chlor	32.8
Chlor	25.0
Chlor	24.2

Figure 11: Binding data (all)

```
ggplot(binding, aes(x = antibiotic, y = binding)) + geom_boxplot()
```

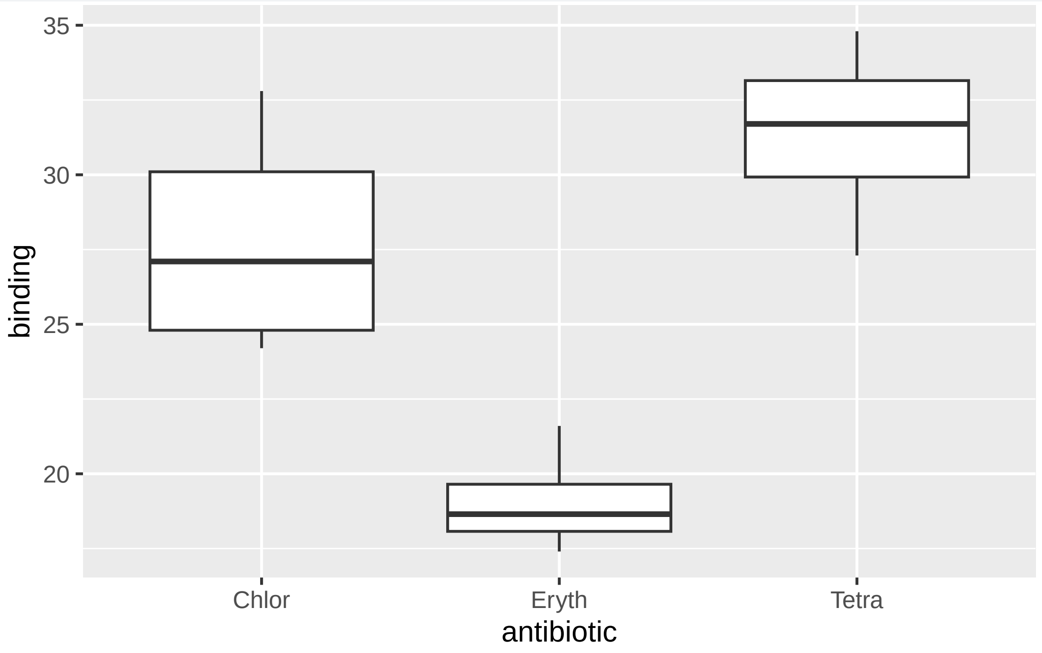


Figure 12: Binding data boxplot

```
bind.1 <- aov(binding ~ antibiotic, data = bind)
summary(bind.1)
```

```
            Df Sum Sq Mean Sq F value   Pr(>F)    
antibiotic   2  320.3  160.13   16.43 0.000991 ***
Residuals    9   87.7    9.75                     
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```
TukeyHSD(bind.1)
```

```
  Tukey multiple comparisons of means
    95% family-wise confidence level

Fit: aov(formula = binding ~ antibiotic, data = bind)

$antibiotic
              diff        lwr       upr     p adj
Eryth-Chlor -8.725 -14.888352 -2.561648 0.0084353
Tetra-Chlor  3.575  -2.588352  9.738352 0.2869496
Tetra-Eryth 12.300   6.136648 18.463352 0.0009040
```

Figure 13: Binding data analysis 1

```
oneway.test(binding ~ antibiotic, data = bind)
```

```

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

data:  binding and antibiotic
F = 23.366, num df = 2.0000, denom df = 5.3673, p-value = 0.002244
```

```
gamesHowellTest(binding ~ factor(antibiotic), data = bind)
```

```

    Pairwise comparisons using Games-Howell test
data: binding by factor(antibiotic)
      Chlor Eryth
Eryth 0.033 -
Tetra 0.400 0.003

P value adjustment method: none
alternative hypothesis: two.sided
```

Figure 14: Binding data analysis 2

```
df1
```

a	b	c
10	14	12
7	9	11

Figure 15: Dataframe `df1`

```
df2
```

gp	y
a	10
b	14
c	12
a	7
b	9
c	11

Figure 16: Dataframe `df2`

```
df3
```

a_lo	a_hi	b_lo	b_hi
10	12	7	9
13	14	6	8

Figure 17: Dataframe `df3`

```
df4
```

gp	level	z
a	lo	10
a	hi	12
b	lo	7
b	hi	9
a	lo	13
a	hi	14
b	lo	6
b	hi	8

Figure 18: Dataframe `df4`

```
df5
```

level	a	b
lo	10	7
hi	12	9
lo	13	6
hi	14	8

Figure 19: Dataframe `df5`

```
df6
```

gp	level	z
a	lo	10
a	hi	12
b	lo	7
b	hi	9

Figure 20: Dataframe `df6` (note that this is just the first four rows of `df4`)

```
df7
```

gp	lo	hi
a	10	12
b	7	9

Figure 21: Dataframe `df7`

```
df8
```

trt	colour	z
u	red	9
v	red	11
u	green	13
w	green	15
v	green	17

```
df8 %>% pivot_wider(names_from = colour, values_from = z)
```

Figure 22: Dataframe `df8` and some code

income	height	weight	age	marital	sex	education	afqt
53000	68	205	53	married	male	15	90.791
5310	62	132	49	married	female	12	78.055
7000	66	180	50	married	female	14	81.344
10000	66	180	53	separated	female	12	67.886
6000	64	165	48	married	female	11	8.653
12000	68	150	50	married	female	12	61.117
25000	66	150	53	divorced	male	11	34.599
83000	62	144	49	divorced	female	12	21.965
160000	71	162	49	divorced	male	16	89.282
40000	73	225	52	married	male	12	4.242
30000	63	160	48	single	female	12	7.495
70000	67	179	51	married	male	12	54.298
62000	62	135	48	married	female	20	40.182
67500	65	190	51	married	female	16	50.223
43000	70	186	49	married	male	12	31.213

Figure 23: Heights and income data (15 randomly chosen rows)

```r
heights.0 <- lm(income ~ height, data = heights)
summary(heights.0)
```

```

Call:
lm(formula = income ~ height, data = heights)

Residuals:
   Min     1Q Median     3Q    Max 
-53876 -24497  -5172  19108 100640 

Coefficients:
            Estimate Std. Error t value Pr(>|t|)   
(Intercept)  -117005      53246  -2.197  0.03034 * 
height          2484        788   3.152  0.00215 **
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 32070 on 98 degrees of freedom
Multiple R-squared:  0.09207,	Adjusted R-squared:  0.0828 
F-statistic: 9.937 on 1 and 98 DF,  p-value: 0.002149
```

Figure 24: Income and height regression

```

Call:
lm(formula = income ~ sex + education + afqt, data = heights)

Residuals:
   Min     1Q Median     3Q    Max 
-60647 -19223  -4109  14528  71140 

Coefficients:
            Estimate Std. Error t value Pr(>|t|)    
(Intercept) -11376.2    15687.3  -0.725 0.470103    
sexfemale   -22783.4     5856.2  -3.890 0.000184 ***
education     4437.2     1298.9   3.416 0.000933 ***
afqt           327.5      119.3   2.745 0.007220 ** 
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 28110 on 96 degrees of freedom
Multiple R-squared:  0.3167,	Adjusted R-squared:  0.2953 
F-statistic: 14.83 on 3 and 96 DF,  p-value: 5.22e-08
```

Figure 25: Regression of income on explanatory variables. The model is called `heights.1`.

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

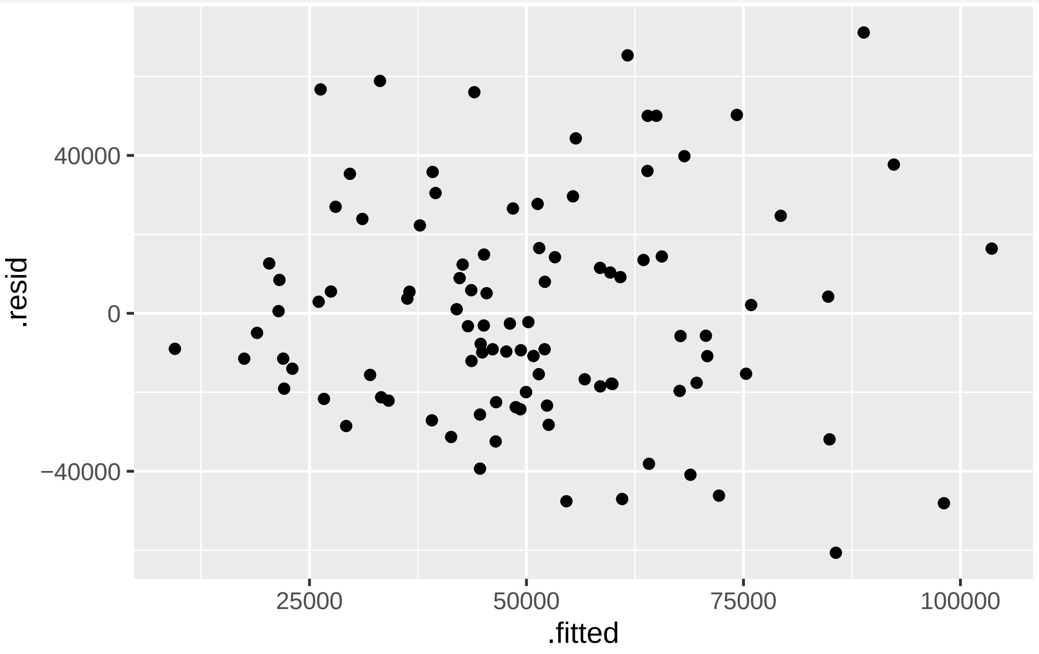


Figure 26: Regression plot 1 of income on explanatory variables

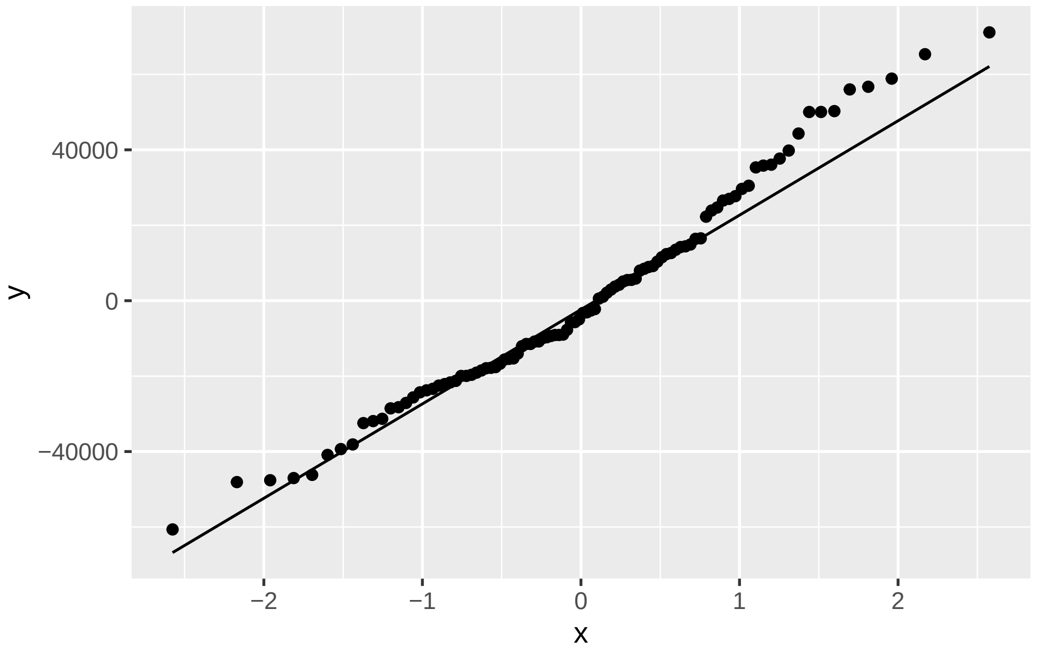


Figure 27: Regression plot 2 of income on explanatory variables

```
boxcox(income ~ sex + education + afqt, data = heights)
```

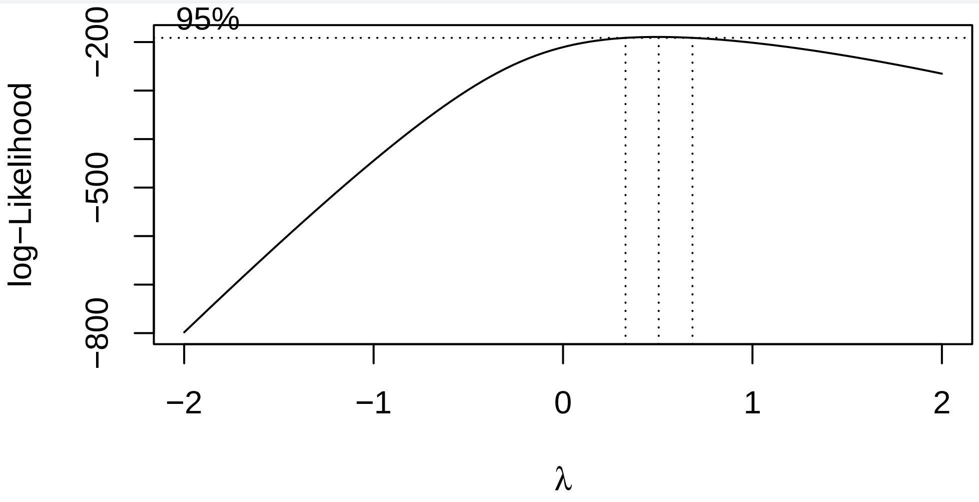


Figure 28: Regression plot 3 of income on explanatory variables

```
variable   mean median   sd  mad     q5    q95 rhat ess_bulk ess_tail
    lp__ -35.59 -35.32 0.73 0.33 -37.03 -35.08 1.00     2092     2612
    p      0.59   0.60 0.07 0.07   0.48   0.70 1.00     1387     2440
```

Figure 29: Stan model results