University of Toronto Scarborough
Department of Computer and Mathematical Sciences
STAC33 (K. Butler), Final Exam
April 25, 2023

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 11 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.
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.

1. A chemical analysis was carried out of 178 Italian wines, from three different cultivars. (A cultivar is a grape plant deliberately bred to have certain desirable characteristics.) The aim of the chemical analysis was to see whether wine made from grapes grown from the three different cultivars differs in any important ways.

   Some of the data file is shown in Figure 2, and the file is saved in your current R Project as `wine.txt`. The dataframe after being read in is shown in Figure 3.

   The questions below ask for Tidyverse R code to accomplish the task described. Explanations are only needed where explicitly asked for.

   (a) [3] Read the data from the file into a dataframe called `wine`. Maximum points are for the simplest method, with a brief explanation of why it will work.

My answer:

The obvious thing is to see that each data value is separated from the next by a comma, and thus the first line here:

```
wine <- read_delim("wine.txt", ",")
```

```
## Rows: 178 Columns: 5
## -- Column specification --------------------------------------------------------
## Delimiter: ","
## chr (1): cultivar_name
## dbl (4): alcohol, malic_acid, ash, mg
##
## i Use `spec()` to retrieve the full column specification for this data.
## i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

```
wine
```

```
## # A tibble: 178 x 5
##    cultivar_name alcohol malic_acid   ash    mg
##    <chr>           <dbl>      <dbl> <dbl> <dbl>
##  1 grignolino       12.9       4.61  2.48    86
##  2 grignolino       13.4       4.6   2.86   112
##  3 barolo           13.9       1.68  2.12   101
##  4 barbera          11.8       0.89  2.58    94
##  5 barolo           13.5       1.8   2.65   110
##  6 barolo           13.0       1.73  2.04    92
##  7 barolo           14.4       1.87  2.38   102
##  8 barolo           13.5       1.81  2.61    96
##  9 barbera          11.9       4.31  2.39    82
## 10 barbera          12         0.92  2        86
## # i 168 more rows
```

This, as you see, works. But it's only two points, because it's not the simplest method. To figure out what that is, think about where you've heard the phrase "comma-separated values" before: this is what the abbreviation `csv` stands for! Hence, `read_csv` ought to read it in as well:

```
wine <- read_csv("wine.txt")
```

```
## Rows: 178 Columns: 5
## -- Column specification --------------------------------------------------------
## Delimiter: ","
## chr (1): cultivar_name
## dbl (4): alcohol, malic_acid, ash, mg
##
## i Use `spec()` to retrieve the full column specification for this data.
## i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

```
wine
```

```
## # A tibble: 178 x 5
##    cultivar_name alcohol malic_acid   ash    mg
##    <chr>           <dbl>      <dbl> <dbl> <dbl>
##  1 grignolino       12.9       4.61  2.48    86
##  2 grignolino       13.4       4.6   2.86   112
##  3 barolo           13.9       1.68  2.12   101
##  4 barbera          11.8       0.89  2.58    94
##  5 barolo           13.5       1.8   2.65   110
##  6 barolo           13.0       1.73  2.04    92
##  7 barolo           14.4       1.87  2.38   102
##  8 barolo           13.5       1.81  2.61    96
##  9 barbera          11.9       4.31  2.39    82
## 10 barbera          12         0.92  2       86
## # i 168 more rows
```

and so it does. This is better because it's simpler, and thus three points for seeing that this will work and why it will work. Using `read_csv` *without* explanation is not enough, because you could have just guessed, and we don't do guesses in this course. So say something about how you know that `read_csv` will work, presumably along the lines of CSV *meaning* values separated by commas.

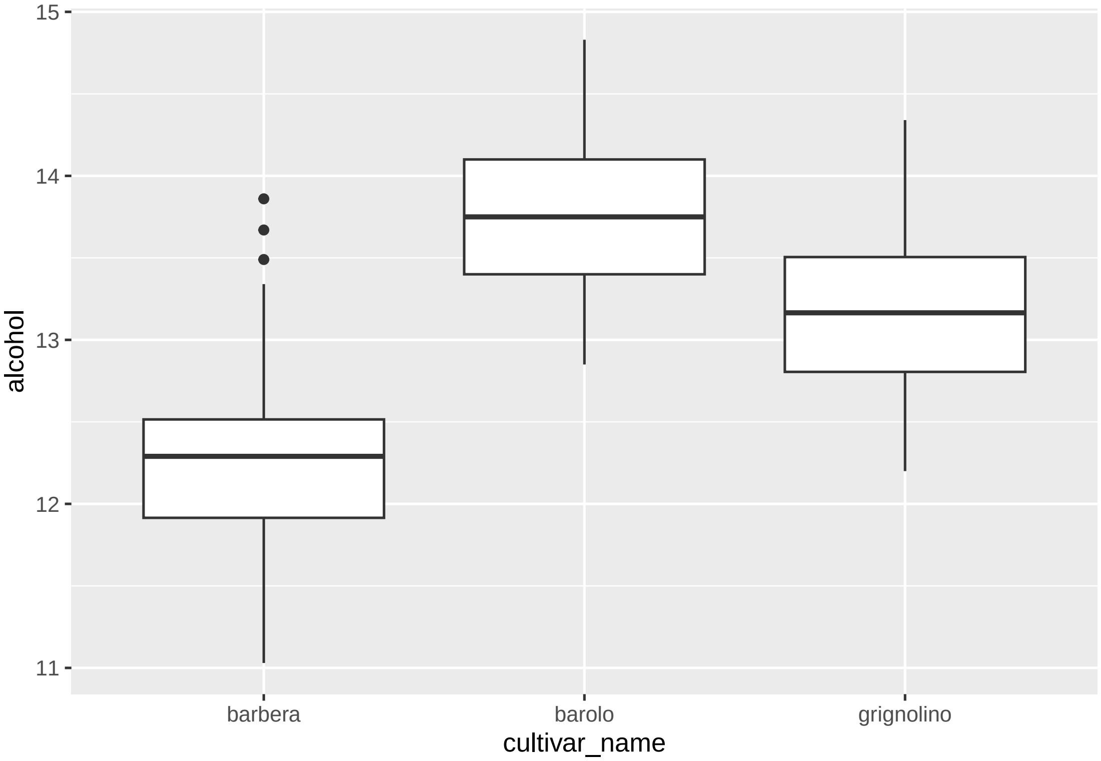
Only one point for `read_csv` without any comment, and three points for that with a clear explanation. (If your explanation gets part of the way towards why it will work, two points.)

(b) [2] Produce a suitable plot of the alcohol content of the wines made from each of the three different cultivars.

My answer:

One categorical variable (`cultivar_name`) and one quantitative one (`alcohol`), so a boxplot:

```
ggplot(wine, aes(x = cultivar_name, y = alcohol)) + geom_boxplot()
```



No explanation needed: if you get this, you know what you are doing and why. Here and below, minus a half point for each small error; a big error might cost you one point.

(c) [3] Calculate the median and inter-quartile range of alcohol content for each of the cultivars.

My answer:

This is a group-by and summarize:

```
wine %>%
  group_by(cultivar_name) %>%
  summarize(alcohol_med = median(alcohol),
            alcohol_iqr = IQR(alcohol))
```

```
## # A tibble: 3 x 3
##   cultivar_name alcohol_med alcohol_iqr
##   <chr>               <dbl>       <dbl>
## 1 barbera              12.3       0.600
## 2 barolo               13.8       0.700
## 3 grignolino           13.2       0.700
```

Give the summaries any names you like. The function for calculating the inter-quartile range has a name that is ALL UPPERCASE.

(d) [2] Display the columns whose names begin with the letter A (uppercase or lowercase), without naming the columns.

My answer:

This one uses a select-helper, in particular `starts_with`:

```
wine %>%
  select(starts_with("a"))
```

```
## # A tibble: 178 x 2
##    alcohol   ash
##      <dbl> <dbl>
##  1    12.9  2.48
##  2    13.4  2.86
##  3    13.9  2.12
##  4    11.8  2.58
##  5    13.5  2.65
##  6    13.0  2.04
##  7    14.4  2.38
##  8    13.5  2.61
##  9    11.9  2.39
## 10    12    2
## # i 168 more rows
```

The select-helpers are not case-sensitive, so you can also do this:

```
wine %>%
  select(starts_with("A"))
```

```
## # A tibble: 178 x 2
##    alcohol   ash
##      <dbl> <dbl>
##  1    12.9  2.48
##  2    13.4  2.86
##  3    13.9  2.12
##  4    11.8  2.58
##  5    13.5  2.65
##  6    13.0  2.04
##  7    14.4  2.38
##  8    13.5  2.61
##  9    11.9  2.39
## 10    12    2
## # i 168 more rows
```

(e) [3] Display all the columns that are quantitative, without naming any of the columns.

My answer:

This uses `where`, because we are selecting columns by a property they have, rather than something to do with their names:

```
wine %>%
  select(where(is.numeric))
```

```
## # A tibble: 178 x 4
##    alcohol malic_acid   ash    mg
##      <dbl>      <dbl> <dbl> <dbl>
##  1    12.9       4.61  2.48    86
##  2    13.4       4.6   2.86   112
##  3    13.9       1.68  2.12   101
##  4    11.8       0.89  2.58    94
##  5    13.5       1.8   2.65   110
##  6    13.0       1.73  2.04    92
##  7    14.4       1.87  2.38   102
##  8    13.5       1.81  2.61    96
##  9    11.9       4.31  2.39    82
## 10    12         0.92  2       86
## # i 168 more rows
```

`is.numeric` is the thing that is TRUE if the column contains numbers, and FALSE otherwise. The following also works, because the data values are decimal numbers, what R knows of as `double` (double-precision floating point numbers):

```
wine %>%
  select(where(is.double))
```

```
## # A tibble: 178 x 4
##    alcohol malic_acid   ash    mg
##      <dbl>      <dbl> <dbl> <dbl>
##  1    12.9       4.61  2.48    86
##  2    13.4       4.6   2.86   112
##  3    13.9       1.68  2.12   101
##  4    11.8       0.89  2.58    94
##  5    13.5       1.8   2.65   110
##  6    13.0       1.73  2.04    92
##  7    14.4       1.87  2.38   102
##  8    13.5       1.81  2.61    96
##  9    11.9       4.31  2.39    82
## 10    12         0.92  2       86
## # i 168 more rows
```

(f) [3] Display the wines that are made from the `barolo` cultivar and that have an ash content of strictly less than 2.2.

My answer:

We now want to display wines, that is, rows, so we need `filter`. To do "and", either have two `filter`s, one after the other:

```
wine %>%
  filter(cultivar_name == "barolo") %>%
  filter(ash < 2.2)
```

```
## # A tibble: 8 x 5
##   cultivar_name alcohol malic_acid   ash    mg
##   <chr>           <dbl>      <dbl> <dbl> <dbl>
## 1 barolo           13.9       1.68  2.12   101
## 2 barolo           13.0       1.73  2.04    92
## 3 barolo           13.0       1.77  2.1    107
## 4 barolo           13.3       1.72  2.14    94
## 5 barolo           14.8       1.64  2.17    97
## 6 barolo           13.4       3.84  2.12    90
## 7 barolo           13.1       1.5   2.1     98
## 8 barolo           13.2       1.78  2.14   100
```

or put the two conditions in the same `filter`, separated by a comma:

```
wine %>%
  filter(cultivar_name == "barolo", ash < 2.2)
```

```
## # A tibble: 8 x 5
##   cultivar_name alcohol malic_acid   ash    mg
##   <chr>           <dbl>      <dbl> <dbl> <dbl>
## 1 barolo           13.9       1.68  2.12   101
## 2 barolo           13.0       1.73  2.04    92
## 3 barolo           13.0       1.77  2.1    107
## 4 barolo           13.3       1.72  2.14    94
## 5 barolo           14.8       1.64  2.17    97
## 6 barolo           13.4       3.84  2.12    90
## 7 barolo           13.1       1.5   2.1     98
## 8 barolo           13.2       1.78  2.14   100
```

(g) [3] Display how many wines of each cultivar have an ash content of strictly less than 2.2. (Do not count any wines with any other ash content.)

My answer:

Find the ones that have a small enough `ash` value first, and then count the cultivar names of these:

```
wine %>%
  filter(ash < 2.2) %>%
  count(cultivar_name)
```

```
## # A tibble: 3 x 2
##   cultivar_name     n
##   <chr>         <int>
## 1 barbera          27
## 2 barolo            8
## 3 grignolino        3
```

Counting also works with `summarize`, but it is one more step, because you have to explicitly `group_by` first:

```
wine %>%
  filter(ash < 2.2) %>%
  group_by(cultivar_name) %>%
  summarize(n = n())
```

```
## # A tibble: 3 x 2
##   cultivar_name     n
##   <chr>         <int>
## 1 barbera          27
## 2 barolo            8
## 3 grignolino        3
```

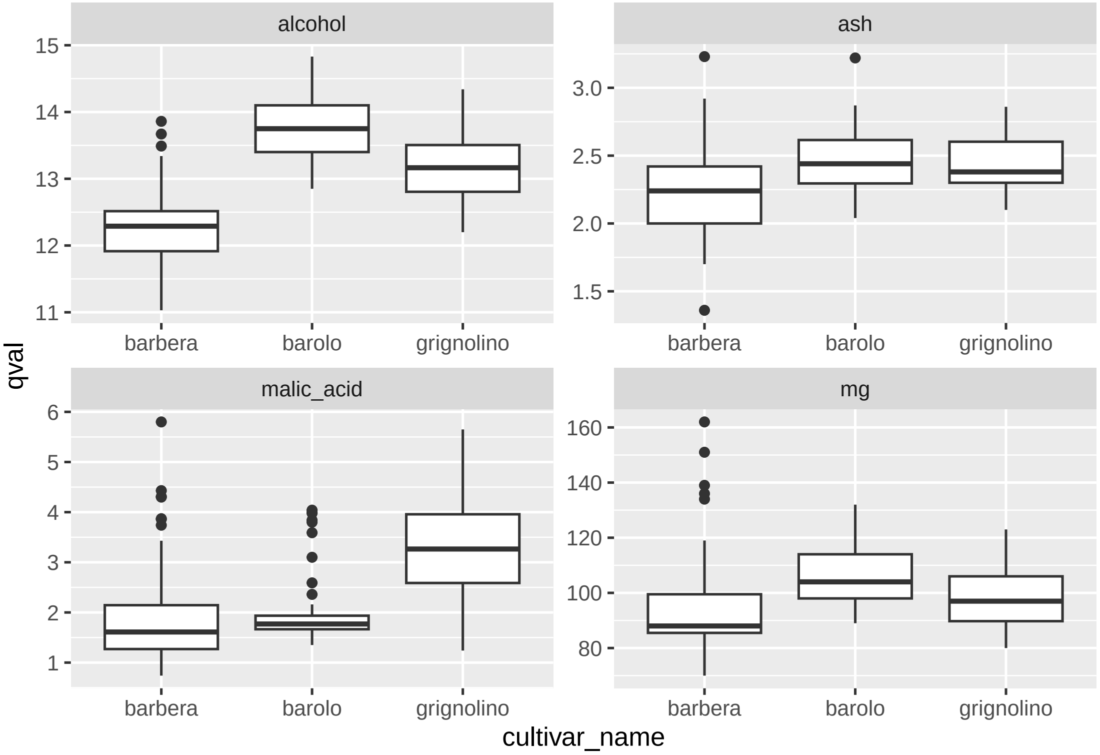
(h) [4] Make a display containing boxplots of each of the quantitative variables for each of the cultivars, using one `ggplot`, and with the side-by-side boxplot for each quantitative variable in a separate sub-plot within the overall display. Bear in mind that some of the quantitative variables have typically bigger values than others.

My answer:

Deliberately challenging. The "sub-plot" thing suggests to use facets, and for that, we need a column with the name of the quantitative variable in it (and the values of all those variables in another column). That in turn suggests pivoting longer. (Or, you might get to this point by remembering that `ggplot` likes long more than wide.)

The idea is the same one as for plotting all the explanatory variables in a regression against the response (or for plotting the residuals against all of the explanatory variables), as in the asphalt example, but this one is boxplots rather than scatterplots:

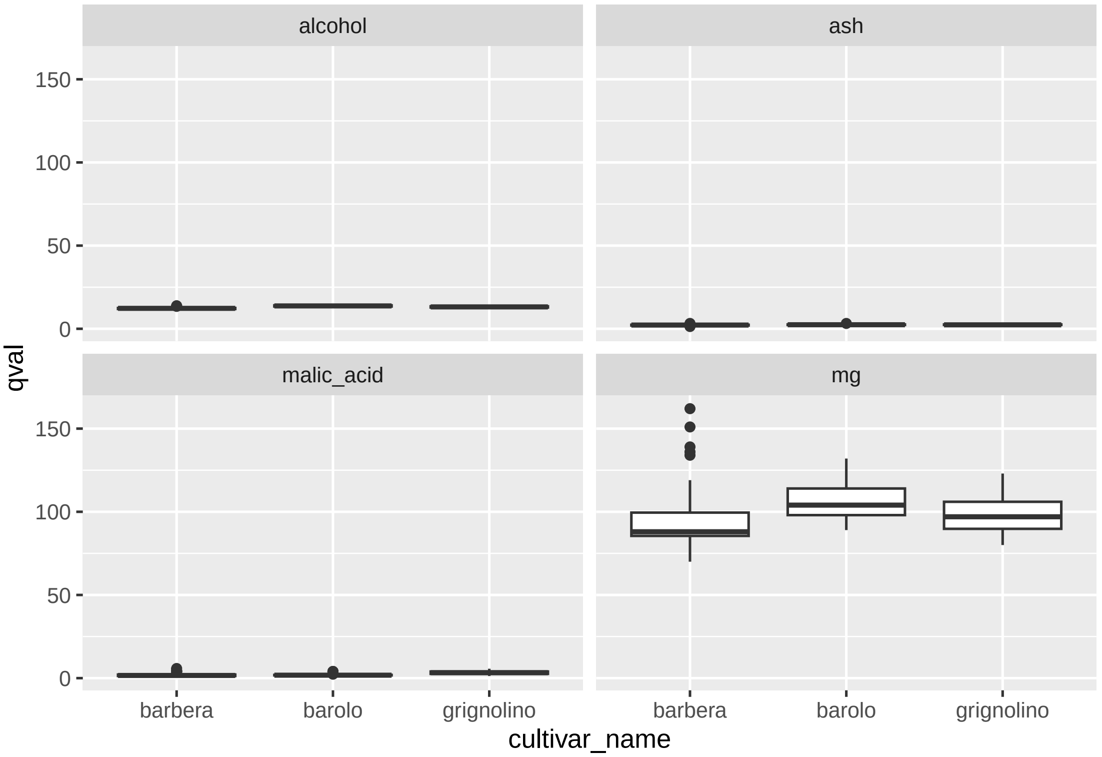
```
wine %>%
  pivot_longer(-cultivar_name, names_to = "qname", values_to = "qval") %>%
  ggplot(aes(x = cultivar_name, y = qval)) + geom_boxplot() +
  facet_wrap(~ qname, scales = "free")
```



You can give your new columns any name you like. I was thinking here of “quantitative name” and “quantitative value”. I think it is easier to choose the columns to pivot longer as “everything but cultivar name”, but use `alcohol:mg` if you prefer.

The `scales = "free"` gives each sub-plot its own scale, which is best because of the last sentence of the question. If you don’t do that, what will happen is this:

```
wine %>%
  pivot_longer(-cultivar_name, names_to = "qname", values_to = "qval") %>%
  ggplot(aes(x = cultivar_name, y = qval)) + geom_boxplot() +
  facet_wrap(~ qname)
```



Only the magnesium content `mg` displays anything like properly; the other values are much smaller in size, and they end up getting compressed down near zero, and you get no sense of their distributions.

Marks: four for the facetted boxplots with their own scales, three if you miss out the `scales = "free"`, two if you do the `pivot_longer` but mess up the graph (there can be a little more if you make some progress towards the graph) or if you successfully produce four separate boxplots, one for any other progress towards a solution, in the grader's estimation.

2. We are planning a study that will use the sign test to test whether the population median is 100 (against a two-sided alternative that it is not equal to 100). The data that our study will provide are assumed to come from a normal distribution with SD 20.

   (a) [2] How do you know that the mean and median of a normal distribution are equal to each other? Explain briefly. You may assume that the mean and *mode* of the normal distribution are equal.

> **My answer:**
>
> The normal distribution is symmetric about the mode, and therefore about the mean. Hence 50% of it is to the left of the mean and 50% to the right, which indicates that the mean and the median are equal (this is the definition of the median).
>
> One point for mentioning symmetry about the mean, and the second for connecting the dots about why a symmetric distribution's mean and median must be the same.
>
> This is really B52 material, but in a course like this, I expect you to be able to work this out even if you have forgotten how you did it before. It is really only symmetry about the mean, plus the definition of the median. (The same thing applies to any symmetric distribution.)
>
> Extra: working out that the *mean* of a symmetric distribution is at the centre of symmetry is a little harder. Call our random variable X, and let the centre of symmetry be m. Write $Y = X - m$; then $E(Y) = E(X) - m$. But Y is symmetric about zero (because X is symmetric about m). This means that Y and $-Y$ have the same distribution, and thus that $E(Y) = E(-Y) = -E(Y)$, and that can only be true if $E(Y) = 0$. Hence $E(X) = m$.
>
> This is the simplest proof I know. You can also write the expectation as an integral and do some careful changes of variable to show that things cancel out (the first change of variable is $y = x - m$, and then you split the integral at m). You can also use properties of odd functions to make the argument.
>
> If you want to work directly with the normal distribution, it is easiest to work with the standard normal, showing that its mean is *zero*, which you undoubtedly proved in B52, and then transform back to the mean you want. I had to be a little more careful in my more general argument, because the distribution might have been bimodal, and the centre of symmetry and the modes would then have been different. But the normal distribution has only one mode, which is also the centre of symmetry.

   (b) [4] We want to estimate, by simulation, the power of the sign test to reject the null hypothesis that the median is 100, against a two-sided alternative, when the median is actually 110, using a sign test, under the conditions given in the question. What code would do that? Hint: the `smmr` package also has a function `pval_sign0` that obtains the two-sided P-value for a sign test. It has two inputs: the null median and a column of data, in that order, and returns only the P-value.

> **My answer:**
>
> This is the usual power by simulation. The process is to generate some random samples from the truth, test the null median, get hold of the P-values, then count how many P-values are less than (or equal to 0.05):

```
tibble(sim = 1:1000) %>%
  rowwise() %>%
  mutate(my_sample = list(rnorm(40, 110, 20))) %>%
  mutate(sign_p = pval_sign0(100, my_sample)) %>%
  count(sign_p <= 0.05)
```

```
## # A tibble: 2 x 2
## # Rowwise:
##   `sign_p <= 0.05`     n
##   <lgl>            <int>
## 1 FALSE              350
## 2 TRUE               650
```

The new thing here is the fourth line, in which we do the sign test and get the P-value in one step.

Extra: This uses a new set of random numbers, so the answer here is slightly different from the power given in the next part.

(c) [3] The code of the previous part gives an estimated power of 0.677. A different power analysis, also applying to this situation, is shown in Figure 4. What code produced this power analysis, bearing in mind what you know so far?

My answer:

This is the kind of output that comes from `power.t.test`. This time, the power is given, so we input that and see what sample size we get:

```
power.t.test(delta = 10, sd = 20, power = 0.677, type = "one.sample",
             alternative = "two.sided")
```

```
##
##      One-sample t test power calculation
##
##               n = 25.38969
##           delta = 10
##              sd = 20
##       sig.level = 0.05
##           power = 0.677
##     alternative = two.sided
```

We have no idea where the sample size of 25.39 came from, so that must be the output, and the 0.677 we were given must be part of the input. Copy the rest of the output to the input of your code. The value of 10 for `delta` is the truth (110) minus the null (100).

(d) [3] According to the information so far, which two tests are we comparing, and which test is the more powerful here? How do you know?

My answer:

We are comparing the sign test (that we did the simulation for), and a one-sample t-test, based on the information in the output, for the same situation but testing the mean instead of the median. (This was the point of showing that the mean and median are equal in this case.)

The t-test is more powerful, because it achieves the same power with a smaller sample size (26 vs. 40).

Extra: this is about the least favourable case for the sign test; we know the t-test must be valid (because the population is normal); the sign test can still be done, but it uses the data much less efficiently (in a case where distance away from the mean/median is actually relevant, because the normal distribution does not have outliers). When the population has long tails, the story can be very different: for example, if the population distribution is "double exponential" (two exponential distributions glued back to back with a mode in the middle), the likelihood of outliers is so large that the sign test can be *more* powerful than the t-test, even though the distribution is symmetric. (The mean can be thrown off a lot more by outliers than the median can.)

3. In an earlier question, we encountered some data from a chemical analysis of some Italian wines. The data, as read into a dataframe `wine`, are shown in Figure 3. In this question, we will see whether the malic acid level differs among wines from the different cultivars.

   (a) [2] Figure 5 shows a plot of malic acid for each cultivar. Why do you think the analyst on this project decided to run a Mood's median test? Explain briefly. (There are two points you need to make).

My answer:

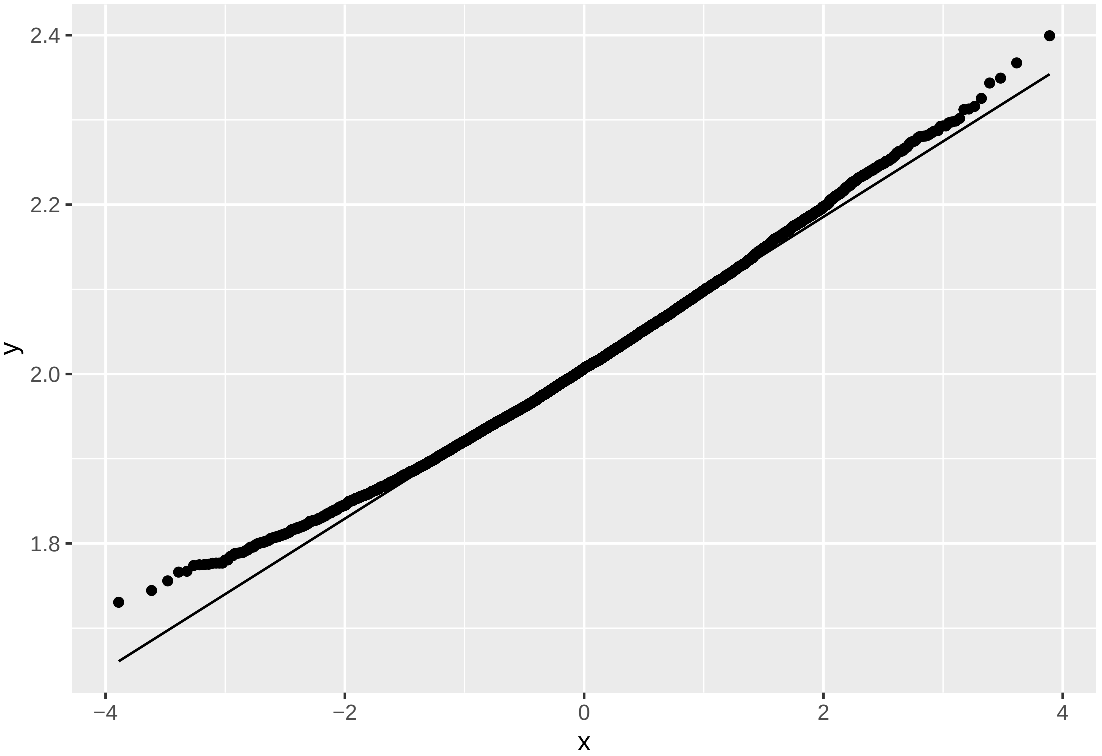
The two points are normality (one point) and sample size (the other). The sample sizes for each group are shown above the plot. Specifically:

- The distribution of malic acid levels are not close to normal for `barolo` (especially) and `barbera`.
- The sample sizes for these two groups are not small: 59 for `barolo` and 71 for `barbera`, but presumably not big enough to overcome the severe skewness / large number of outliers.

Since we doubt that we have sufficient normality in all three groups, we prefer to run a Mood's median test to compare the malic acid levels. (All you have to do is to find *one* cultivar for which the distribution of malic acid levels is not normal enough given the sample size.)

Extra: I have the means to do a bootstrap sampling distribution of the sample mean. Here is `barolo`, the worst one:

```r
wine %>% filter(cultivar_name == "barolo") -> barolo
tibble(sim = 1:10000) %>%
  rowwise() %>%
  mutate(my_sample = list(sample(barolo$malic_acid, replace = TRUE))) %>%
  mutate(my_mean = mean(my_sample)) %>%
  ggplot(aes(sample = my_mean)) + stat_qq() + stat_qq_line()
```



This, even with a sample of size 59, is still skewed to the right (because of the long right tail / high outliers), though you might reasonably say that this is better than you would expect.

(b) [2] Mood's median test is run on these data, as shown in Figure 6. What do you conclude from this Figure, in the context of the data?

My answer:

Our null hypothesis is that the median malic acid content is the same for wines made from all three cultivars. This null hypothesis is very clearly rejected (P-value about 10^{-9}), and so we reject the null and conclude that the median malic acid content values are not all the same (or at least one of them is different).

This is as far as you go from this output. If you want to say something about which cultivars differ from which, you need the pairwise median tests, which are in the next Figure, not this one.

(c) [3] Are we justified in looking at Figure 7 for these data? Explain briefly why or why not. If appropriate, what do we conclude from this Figure?

My answer:

We are justified in looking at the Figure because the Mood's median test was significant, and there are some differences between cultivars to find.

Both differences involving `grignolino` are significant, but the correct P-value for comparing `barbera` and `barolo` is 0.119 (the adjusted one from the right column, adjusted for doing three tests at once), so these do not differ in median malic acid level.

(The boxplot in Figure 5 indicates that `grignolino` has a *higher* median malic acid content than the others.)

(d) [3] Some mystery code is shown in Figure 8. What task does running this code accomplish, and why might you be interested in the results? Hint: you are looking for the overall purpose of running the code, *not* a line-by-line description of what the code does. We are working at a higher level than that.

My answer:

This obtains a bootstrap sampling distribution of the sample mean (half a point) of malic acid levels (half a point) for wines from the `barolo` cultivar (half a point), and draws a normal quantile plot of the results (half a point). Make sure you say that it's for malic acid, and only for wines from this cultivar.

We would be interested in the results because if the plot looks normally distributed (we saw in an Extra earlier that in fact it does not), we could run a regular or Welch ANOVA instead, even though the distribution of malic acid values for this cultivar looks not at all normal (and in that case, the sample size would be big enough to overcome the skewness). For the last point, say something about why normality would influence what we do.

4. In each part of this question, you will be given a dataframe (shown in a Figure). You will either be given some code and asked what output it will produce, or you will be given the output and asked what code will produce it.

   (a) [3] Given the dataframe d1 shown in Figure 9, and the code in Figure 10, what output will be produced?

My answer:

In all of these, the order of the columns and rows is not important, as long as the right values are in the same row, and the right columns exist.

This:

```
d1 %>% pivot_longer(-id, names_to = "name", values_to = "value")
```

```
## # A tibble: 4 x 3
##      id name  value
##   <dbl> <chr> <dbl>
## 1     1 a        10
## 2     1 b        11
## 3     2 a         8
## 4     2 b         9
```

   (b) [3] A dataframe d2 is shown in Figure 11. Some output from operating on d2 is shown in Figure 12. What code would produce that output, starting from d2? Hint: the start of your code should be d2 %>%.

My answer:

The best answer is this:

```
d2 %>% pivot_longer(-row, names_to = c("gender", "what"),
                    names_sep = "_", values_to = "measure")
```

```
## # A tibble: 8 x 4
##     row gender what  measure
##   <dbl> <chr>  <chr>   <dbl>
## 1     7 m      ht        180
## 2     7 f      ht        150
## 3     7 m      wt         80
## 4     7 f      wt         60
## 5     8 m      ht        185
## 6     8 f      ht        160
## 7     8 m      wt         90
## 8     8 f      wt         55
```

using two things in `names_to`. Three points for that, or for any other way that will select the columns you want (for example, the select-helper `contains("_")`, or just `m_ht:f_wt`).

A slightly inferior method is to recognize that a standard `pivot_longer` will produce almost the right thing:

```
d2 %>% pivot_longer(-row, names_to = "x", values_to = "measure")
```

```
## # A tibble: 8 x 3
##     row x     measure
##   <dbl> <chr>   <dbl>
## 1     7 m_ht      180
## 2     7 f_ht      150
## 3     7 m_wt       80
## 4     7 f_wt       60
## 5     8 m_ht      185
## 6     8 f_ht      160
## 7     8 m_wt       90
## 8     8 f_wt       55
```

but then the column I called `x` here will need `separate`-ing out:

```
d2 %>% pivot_longer(-row, names_to = "x", values_to = "measure") %>%
  separate(x, into = c("gender", "what"))
```

```
## # A tibble: 8 x 4
##     row gender what  measure
##   <dbl> <chr>  <chr>   <dbl>
## 1     7 m      ht        180
## 2     7 f      ht        150
## 3     7 m      wt         80
## 4     7 f      wt         60
## 5     8 m      ht        185
## 6     8 f      ht        160
## 7     8 m      wt         90
## 8     8 f      wt         55
```

Two points for this, since the first version is more concise, and shows that you have learned more about how `pivot_longer` works.

(c) [3] Given the dataframe `d3` shown in Figure 13, and the code in Figure 14, what output will be produced?

My answer:

This:

```
d3 %>% pivot_longer(-id, names_to = c(".value", "col"), names_sep = "_")
```

```
## # A tibble: 4 x 3
##      id col       x
##   <dbl> <chr> <dbl>
## 1     4 1        10
## 2     4 2        11
## 3     6 1         8
## 4     6 2         9
```

The column names in `d3` have `x` in their names, so the `.value` will make a new column *called* `x`. The rest of the column names in `d3`, after the underscore, will go into a column called `col`.

(d) [2] Given the dataframe `d4` shown in Figure 15, and the code in Figure 16, what output will be produced?

My answer:

This:

```
d4 %>% separate(x, into = c("gender", "what"), sep = "_")
```

```
## # A tibble: 4 x 4
##     row gender what  measure
##   <dbl> <chr>  <chr>   <dbl>
## 1     7 m      ht        180
## 2     7 f      ht        150
## 3     7 m      wt         80
## 4     7 f      wt         60
```

This is a pretty easy one to guess.

(e) [3] A dataframe `d5` is shown in Figure 17. Some output from operating on `d5` is shown in Figure 18. What code would produce that output, starting from `d5`?

My answer:

```
d5 %>% pivot_wider(names_from = group, values_from = x)
```

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

This is meant to be a straightforward application of `pivot_wider`.

(f) [4] Given the dataframe `d6` shown in Figure 19, and the code in Figure 20, what output will be produced?

My answer:

```
d6 %>% pivot_wider(names_from = z, values_from = y)
```

```
## # A tibble: 3 x 4
##   x       low  high medium
##   <chr> <dbl> <dbl>  <dbl>
## 1 c        16    NA     NA
## 2 b        22    18     NA
## 3 a        NA    NA     20
```

There are too few data values to fill the widened dataframe: there will be three columns (the values in `z`) and three rows (the values in `x`), but there are only four values in `y`, so five of the cells will be missing. A hint for working these out is to make a blank dataframe whose columns are the different things in `names_from`. For the rows, the name of the column is anything not mentioned in the `pivot_wider` (that is to say, `x` here) and the values of the rows are the *distinct* values in that column, that is, `c`, `b` and `a`. That gives you an empty dataframe with rows and columns, and then you take the values of `y` and put them in the right row and column according to their values of `x` and `z`. Finally, you put an `NA` in any cells that are empty at the end of this.

Sometimes two different values of the `values_from` variable have to go in the same row and column, and then you get a warning and list-columns (which happened to me the first time I tried to devise this question). But I didn't want to give you one of those, so I changed things around.

The grader will check that you have the four values 16, 18, 20, and 22 in the right places (row according to `x` and column according to `z`), and missings everywhere else. The order of rows or columns does not matter. This one is four marks because there is a lot that can go wrong, and the grader will make a call about how close you got to the right answer.

5. In 1982, a researcher named Engel studied the relationship between income and food expenditure in Belgium, with the aim of seeing whether food expenditure changed as income increased. 234 working-class households were studied, and annual income and food expenditure were measured in Belgian francs. Some of the data are shown in Figure 21.

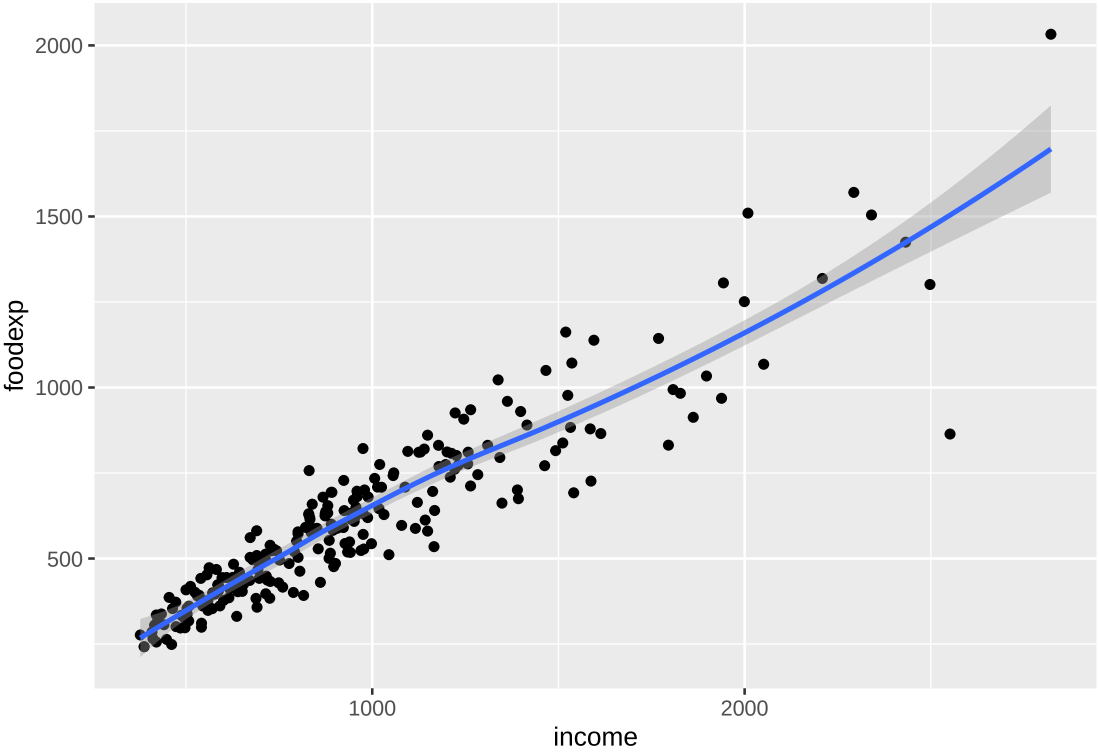
   (a) [2] What code would draw a suitable plot of these data, with, if appropriate for your plot, a smooth trend?

My answer:

Two quantitative variables implies a scatterplot. The question says that income is explanatory and food expenditure is the response (which you can also see from looking ahead to the next part), so:

```
ggplot(engel, aes(x = income, y = foodexp)) + geom_point() + geom_smooth()
```

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



Add a smooth trend to your scatterplot rather than a regression line, because we don't know yet whether the relationship is straight. This graph says that it is (there is not an obvious curve) so fitting a linear regression (as in the next part) is the obvious thing to try.

(b) [3] A regression analysis with its output is shown in Figure 22. According to this Figure, is there a significant effect of income on food expenditure, and, from this analysis, is the trend upward, downward, or non-existent? Explain briefly in each case.

My answer:

The P-value on the `income` line (or of the F-statistic at the bottom of the output) is very small (less than 2×10^{-16}), so there is a significant effect of income on food expenditure. Two points.

The slope of income, 0.546, is positive, so the trend is (significantly) upward. One point.

I said "significant effect of income" in the question, so you need to supply a P-value and reject a null hypothesis that there is no effect. Telling me, for example, that R-squared is high doesn't answer the question as asked. (Statistically) significant means that you need to do a test. One out of two if you answer the question about significance this way (otherwise) correctly.

You might *guess* that a household with more income will spend more on food, but the point of this question is to see whether it is supported by the data, and the regression analysis says that it is.

We are about to critique this analysis, but the point of this question was to make sure that you know what "significant" means, and what the slope signifies.

(c) [2] A plot of residuals against fitted values is shown in Figure 23, for the regression shown in Figure 22. What is the major problem with this residual plot? Explain briefly (that is, say how you know).

My answer:

The major problem is one of "fanning out": that is to say, the residuals are larger in size on the right than they are on the left, rather than being of about the same size all the way across. Another way to say this is that food expenditure is estimated less accurately when income is higher.

Extra: you might imagine that food expenditure could typically be a *percentage* of household income, and so when income is higher, the actual food expenditure could vary more without noticeably changing that percentage. This would favour taking logs, which (as you'll see) is not what happens in fact.

(d) [2] In an attempt to fix the problem, a Box-Cox analysis is run, as shown in Figure 24. What do you conclude from this Figure?

My answer:

The only "round" value of λ within the 95% confidence interval is 0.5, so the best transformation is to use the square root of food expenditure in a regression instead of food expenditure itself.

"Take the square root" is not a complete answer: square root of what?

(e) [2] A regression that uses the results of the Box-Cox analysis is fitted, and the residuals vs. fitted values are plotted in Figure 25. The statistician on the project knows that there was one family whose food expenditure was much less than you would expect from their income. Aside from this, do you think the residual plot is satisfactory now? Explain briefly.

My answer:

The observation mentioned in the question is the point at the bottom right of Figure 25, with the most negative residual. Aside from that, the other observations form a nice random horizontal band that is no longer getting wider as income increases. Thus, the residual plot is satisfactory now apart from that one observation.

6. A debt is money that is owed to another person or company that has not yet been paid. In the past, being in debt was seen as a bad thing, but attitudes towards debt have changed over time, and borrowing money to pay for a large purchase like a house or a car or an education can now be seen as being financially responsible.

   An economist designed a survey in which they tried to find out how attitudes towards debt are influenced by other things such as age, money-management skills, or "locus of control". This last is a concept from psychology. A person can have an "external locus of control", when they feel that most of the things that happen to them are caused by outside events, or an "internal locus of control", where most of the things that happen to them are caused by things inside the person (such as decisions they made).

   A complete list of the survey items is shown in Figure 26. All variables are quantitative (and are treated as such here), even though many of them are more ordinal than quantitative (scores on a scale). Some of the items have a yes/no answer; in those cases, 1 means "yes" and 0 means "no". After removing surveys with missing answers, there were complete surveys from 304 respondents. Some of the data is shown in Figure 27.

   (a) [3] A regression is fitted predicting attitudes to debt from all the other variables. This is shown in Figure 28. Since there were a lot of explanatory variables, I decided to remove the five variables shown in the `update` line at the top of Figure 29; the resulting regression output is shown in the remainder of that Figure. Finally, I ran the test shown in Figure 30.

   Why was it necessary to run the test shown in Figure 30, and what do you conclude from it in the context of the data?

   My answer:

   It was necessary to run the test shown in the last Figure because I removed more than one, actually five, explanatory variables all at once. The *t*-tests in regression `debt.1` only apply if I am taking explanatory variables out one at a time, as in backward elimination (or the use of `step`). One point.

   The null hypothesis is that the two models being compared fit equally well (with the alternative being that the bigger model `debt.1` fits better). The null is not rejected here. One more point for saying that, or for otherwise making it clear that you know this.

   The last point is to say that we should therefore go with model `debt.2`, and that taking out all those explanatory variables was a good thing to do, because they did not add anything to the model.

   (b) [2] What other comparison of models `debt.1` and `debt.2` (that is, not a hypothesis test) suggests that the conclusion from your test in the previous part makes sense?

   My answer:

   "Not a test" suggests that you might want to compare a measure of fit like R-squared. You can go two ways with this:

   - The decrease in R-squared from `debt.1` to `debt.2` is very small, from 0.2043 to 0.1926, so there is almost no difference in fit.

- The *adjusted* R-squared actually *increased* slightly from `debt.1` to `debt.2` (from 0.1715 to 0.1735), indicating that the smaller model actually fits *better* when you allow for its complexity. (I'm not sure I talked about adjusted R-squared in class this time, but if you saw it on the video or learned about it in C67, you can certainly talk about it here.)

(c) [3] For the model `debt.2`, Figure 31 shows a plot of residuals against fitted values, and Figure 32 shows a normal quantile plot of residuals for the same model. Comment briefly on each of these two plots, and make an overall comment considering the two plots together.

My answer:

- The residuals vs fitted value plot is pretty much a textbook example of randomness with no pattern whatever. (The only quibble might be the most negative residual at the bottom. This is not really an outlier, as I comment further below.)
- The normal quantile plot indicates that the residuals have close to a normal distribution, with no real evidence of non-normality, because the points are all close to the line. (That most negative residual appears at the bottom left of this plot; it is noticeably more negative than the other residuals, but not unusually negative compared to a normal distribution because it is close to the line. Hence it does not indicate any sort of problem.)
- Overall, on the evidence here, the regression `debt.2` is completely satisfactory.

One point for each of those.

(d) [2] What other plot or plots would you like to see to confirm your impressions about the model `debt.2`?

My answer:

The standard other plots in a multiple regression are of residuals against each of the explanatory variables, so ask for those.

Extra: there are rather a lot of them, so I didn't make them all to show you, but it was reasonable for you to ask for them (and if you were doing this analysis yourself, you could certainly be expected to make them).

You might ask for residuals vs. the explanatory variables that are in the model, or you might ask for residuals against *all* the explanatory variables, including the ones we removed from `debt.1`. The advantage to looking at all of them is that you might be alerted to some explanatory variables that have a curved relationship with the response `prodebt`, but that did not have a significant linear relationship because the trend was (say) up and then down again.

For this question, you can ask for plots of residuals against the explanatory variables that are in the model `debt.2`, or all the explanatory variables, or against "the explanatory variables" without making it clear which set of them you mean. Any of those is good.

(e) [2] Two of the explanatory variables in the model `debt.2` are `agegp` and `ccarduse`. Based on what you know or can guess, does the *sign* of the slope estimate for each variable (that is, whether it is positive or negative) make sense? Explain briefly.

My answer:

Look back at Figure 29:

- `agegp` has a negative slope. This means that older people (with a higher values of `agegp`) are *less* favourable towards debt. Make some sort of comment about older people liking to buy things outright more than younger people (or, if you like, coming from an era where it was *possible* to do that). Or refer back to the question and say that older people come from a time where debt was considered bad.
- `ccarduse` is how often the respondent uses credit cards. This has a positive slope. If this is higher, meaning the respondent uses credit cards "regularly" as opposed to "never", they also have a more favourable attitude towards debt. This is not very surprising, because every time you use a credit card, you are actually putting yourself in debt (until the credit card bill comes), so somebody who likes to use a credit card had better be comfortable with the idea of debt.

One point for saying whether the slope is positive or negative along with an explanation of why it makes sense, in each case. Saying the slope is positive (or negative) by itself is not worth anything. To get any credit, you have to talk about what it means.

7. The exponential distribution has (continuous) density function $f(x) = \beta e^{-\beta x}$ for $x \geq 0$, where β is a non-negative parameter. The distribution has mean $1/\beta$. Our aim in this problem is to estimate β using Bayesian methods. Stan has an `exponential` distribution with one input (that is β), and also a `uniform` distribution, whose two inputs are the lower and upper limits of that distribution.

   (a) [2] Before looking at any data, we think that the mean μ is almost certainly between 2 and 10. What does this tell us about β, before looking at any data?

   My answer:

   If $2 \leq \mu \leq 10$, then $2 \leq 1/\beta \leq 10$ and thus $1/10 \leq \beta \leq 1/2$, that is, β is almost certainly between 0.1 and 0.5.

   If you say something is between a and b, a had better be less than b. So saying that β is between 1/2 and 1/10 is sloppy, and will cost you a half point.

   Extra: You might have seen another parametrization of the exponential distribution, with density $(1/\alpha)e^{-x/\alpha}$; written this way, the mean is α. But I wanted to give you one where the mean and the data values did not have an obvious association. Besides, Stan's `exponential` distribution has a β-style parametrization, so that's what you'll need to work with.

   (b) [3] Write the `model` section of a Stan program to estimate β, assuming a uniform prior between the two limits you found above, and calling your data `x`.

   My answer:

   Adapt the one in the lecture (or in worksheet 11): the prior and the likelihood:

```
model {
  // prior
  beta ~ uniform(0.1, 0.5);
  // likelihood
  x ~ exponential(beta);
}
```

   The comments are optional but a good idea. Minus a half point here for forgetting the semicolons, but no further deductions for forgetting them below.

   If you didn't get (a), use whatever you had there in your `uniform`. If you didn't have any numbers, the right strategy is to call them something, like u and v, and say "where u and v are the answers to (a)". If you don't make any additional errors in (b), you get full marks for that. Expect me to check back to make sure you are being consistent.

(c) [3] Add appropriate sections to your Stan code to make a complete Stan program. You can assume that there will be 10 observations in your data, and that the observations will be decimal numbers.

My answer:

There needs to be a `data` section and a `parameters` section. Once again, you can adapt ones you have seen before:

```
data {
  real x[10];
}

parameters {
  real<lower=0> beta;
}
```

For `beta`, since you have a uniform prior, the posterior is also between those same limits (as I mention below), so you can use those limits on `beta` here as well. If you don't understand that point and just copied the limits over, you may have gotten lucky.

The data will be decimal numbers, so the `x` in the `data` section now has to be `real` (rather than `int` as in the lecture example). In the description of the exponential distribution in the question, I said that β had to be non-negative, so that has to show up here.

Some people called their data `y` in the previous part, which was not what I asked for (and so was penalized there), but if you have consistently used `y` for your data, you are all right here. (I checked. I am willing to put up with a lot as long as you are consistent.)

The data are also non-negative, so you can also put a `<lower=0>` before the `x[10]`. The only effect this actually has is to have Stan check that your data values are all non-negative and to throw an error if they are not, but you might (reasonably) say that this is a valuable thing to be checking as well.

One point each for a sensible `data` and a sensible `parameters`. Half a point for getting the basic idea right but missing something important.

These two sections go above the `model` section that you wrote before. The third point for saying that or otherwise implying it, like writing something like this below what you have here:

```
model {
  ....
}
```

I asked for you to *add* to what you had before, to make a *complete* Stan program, so your answer needs to be something that will compile, and just the `data` and `parameters` sections won't.

Stan actually has only `real` and `int` as data types; there is nothing like `double` (use `real`). But `double` shows the right thinking.

(d) [2] Assuming that your Stan code has been saved in a file `expo.stan`, what R code would compile it to C++?

My answer:

This:

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

There is a warning that is because of the way we learned to declare `x`. Save it wherever you like. I'm not picky about the name `expo`, though I think it's a pretty good one.

Extra: There is a more modern way of declaring arrays that I can never remember, so this way is in the notes and still works, at least for the time being.

Added later: the right way is apparently this:

```
array[10] real<lower=0> x;
```

I have to update the notes with this for next year, because at some point the way we learned is going to stop working (that is what "deprecated" means).

If you do it this way (earlier), you really ought to say where you found out about it from, but I will accept this here without comment (because it is better than my way).

(e) [3] Some data is shown in Figure 33, stored in a variable `w`. What R code will set up this data suitably, and draw random samples from the posterior distribution of β for the data in `w`?

My answer:

There is a little care that needs to be taken, because our data in our Stan program is called `x`, so we have to make sure the data that finds its way there is also called `x`:

```
expo_data <- list(x = w)
```

and then

```
expo_fit <- expo$sample(data = expo_data)
```

```
## Running MCMC with 4 sequential chains...
##
## Chain 1 Iteration:    1 / 2000 [  0%]  (Warmup)
## Chain 1 Iteration:  100 / 2000 [  5%]  (Warmup)
## Chain 1 Iteration:  200 / 2000 [ 10%]  (Warmup)
## Chain 1 Iteration:  300 / 2000 [ 15%]  (Warmup)
## Chain 1 Iteration:  400 / 2000 [ 20%]  (Warmup)
## Chain 1 Iteration:  500 / 2000 [ 25%]  (Warmup)
## Chain 1 Iteration:  600 / 2000 [ 30%]  (Warmup)
## Chain 1 Iteration:  700 / 2000 [ 35%]  (Warmup)
## Chain 1 Iteration:  800 / 2000 [ 40%]  (Warmup)
## Chain 1 Iteration:  900 / 2000 [ 45%]  (Warmup)
## Chain 1 Iteration: 1000 / 2000 [ 50%]  (Warmup)
## Chain 1 Iteration: 1001 / 2000 [ 50%]  (Sampling)
## Chain 1 Iteration: 1100 / 2000 [ 55%]  (Sampling)
## Chain 1 Iteration: 1200 / 2000 [ 60%]  (Sampling)
## Chain 1 Iteration: 1300 / 2000 [ 65%]  (Sampling)
## Chain 1 Iteration: 1400 / 2000 [ 70%]  (Sampling)
## Chain 1 Iteration: 1500 / 2000 [ 75%]  (Sampling)
## Chain 1 Iteration: 1600 / 2000 [ 80%]  (Sampling)
## Chain 1 Iteration: 1700 / 2000 [ 85%]  (Sampling)
## Chain 1 Iteration: 1800 / 2000 [ 90%]  (Sampling)
## Chain 1 Iteration: 1900 / 2000 [ 95%]  (Sampling)
## Chain 1 Iteration: 2000 / 2000 [100%]  (Sampling)
## Chain 1 finished in 0.0 seconds.

## Chain 2 Rejecting initial value:

## Chain 2   Log probability evaluates to log(0), i.e. negative infinity.

## Chain 2   Stan can't start sampling from this initial value.

## Chain 2 Iteration:    1 / 2000 [  0%]  (Warmup)
## Chain 2 Iteration:  100 / 2000 [  5%]  (Warmup)
## Chain 2 Iteration:  200 / 2000 [ 10%]  (Warmup)
## Chain 2 Iteration:  300 / 2000 [ 15%]  (Warmup)
## Chain 2 Iteration:  400 / 2000 [ 20%]  (Warmup)
## Chain 2 Iteration:  500 / 2000 [ 25%]  (Warmup)
## Chain 2 Iteration:  600 / 2000 [ 30%]  (Warmup)
## Chain 2 Iteration:  700 / 2000 [ 35%]  (Warmup)
## Chain 2 Iteration:  800 / 2000 [ 40%]  (Warmup)
## Chain 2 Iteration:  900 / 2000 [ 45%]  (Warmup)
## Chain 2 Iteration: 1000 / 2000 [ 50%]  (Warmup)
## Chain 2 Iteration: 1001 / 2000 [ 50%]  (Sampling)
## Chain 2 Iteration: 1100 / 2000 [ 55%]  (Sampling)
## Chain 2 Iteration: 1200 / 2000 [ 60%]  (Sampling)
## Chain 2 Iteration: 1300 / 2000 [ 65%]  (Sampling)
## Chain 2 Iteration: 1400 / 2000 [ 70%]  (Sampling)
## Chain 2 Iteration: 1500 / 2000 [ 75%]  (Sampling)
## Chain 2 Iteration: 1600 / 2000 [ 80%]  (Sampling)
## Chain 2 Iteration: 1700 / 2000 [ 85%]  (Sampling)
## Chain 2 Iteration: 1800 / 2000 [ 90%]  (Sampling)
## Chain 2 Iteration: 1900 / 2000 [ 95%]  (Sampling)
## Chain 2 Iteration: 2000 / 2000 [100%]  (Sampling)
## Chain 2 finished in 0.1 seconds.
## Chain 3 Iteration:    1 / 2000 [  0%]  (Warmup)
## Chain 3 Iteration:  100 / 2000 [  5%]  (Warmup)
## Chain 3 Iteration:  200 / 2000 [ 10%]  (Warmup)
## Chain 3 Iteration:  300 / 2000 [ 15%]  (Warmup)
## Chain 3 Iteration:  400 / 2000 [ 20%]  (Warmup)
## Chain 3 Iteration:  500 / 2000 [ 25%]  (Warmup)
## Chain 3 Iteration:  600 / 2000 [ 30%]  (Warmup)
## Chain 3 Iteration:  700 / 2000 [ 35%]  (Warmup)
## Chain 3 Iteration:  800 / 2000 [ 40%]  (Warmup)
## Chain 3 Iteration:  900 / 2000 [ 45%]  (Warmup)
## Chain 3 Iteration: 1000 / 2000 [ 50%]  (Warmup)
## Chain 3 Iteration: 1001 / 2000 [ 50%]  (Sampling)
## Chain 3 Iteration: 1100 / 2000 [ 55%]  (Sampling)
## Chain 3 Iteration: 1200 / 2000 [ 60%]  (Sampling)
## Chain 3 Iteration: 1300 / 2000 [ 65%]  (Sampling)
## Chain 3 Iteration: 1400 / 2000 [ 70%]  (Sampling)
## Chain 3 Iteration: 1500 / 2000 [ 75%]  (Sampling)
## Chain 3 Iteration: 1600 / 2000 [ 80%]  (Sampling)
## Chain 3 Iteration: 1700 / 2000 [ 85%]  (Sampling)
## Chain 3 Iteration: 1800 / 2000 [ 90%]  (Sampling)
## Chain 3 Iteration: 1900 / 2000 [ 95%]  (Sampling)
## Chain 3 Iteration: 2000 / 2000 [100%]  (Sampling)
## Chain 3 finished in 0.1 seconds.

## Chain 4 Rejecting initial value:

## Chain 4   Log probability evaluates to log(0), i.e. negative infinity.

## Chain 4   Stan can't start sampling from this initial value.

## Chain 4 Rejecting initial value:

## Chain 4   Log probability evaluates to log(0), i.e. negative infinity.

## Chain 4   Stan can't start sampling from this initial value.

## Chain 4 Rejecting initial value:

## Chain 4   Log probability evaluates to log(0), i.e. negative infinity.

## Chain 4   Stan can't start sampling from this initial value.

## Chain 4 Rejecting initial value:

## Chain 4   Log probability evaluates to log(0), i.e. negative infinity.

## Chain 4   Stan can't start sampling from this initial value.

## Chain 4 Iteration:    1 / 2000 [  0%]  (Warmup)
## Chain 4 Iteration:  100 / 2000 [  5%]  (Warmup)
## Chain 4 Iteration:  200 / 2000 [ 10%]  (Warmup)
## Chain 4 Iteration:  300 / 2000 [ 15%]  (Warmup)
## Chain 4 Iteration:  400 / 2000 [ 20%]  (Warmup)
## Chain 4 Iteration:  500 / 2000 [ 25%]  (Warmup)
## Chain 4 Iteration:  600 / 2000 [ 30%]  (Warmup)
## Chain 4 Iteration:  700 / 2000 [ 35%]  (Warmup)
## Chain 4 Iteration:  800 / 2000 [ 40%]  (Warmup)
## Chain 4 Iteration:  900 / 2000 [ 45%]  (Warmup)
## Chain 4 Iteration: 1000 / 2000 [ 50%]  (Warmup)
## Chain 4 Iteration: 1001 / 2000 [ 50%]  (Sampling)
## Chain 4 Iteration: 1100 / 2000 [ 55%]  (Sampling)
## Chain 4 Iteration: 1200 / 2000 [ 60%]  (Sampling)
## Chain 4 Iteration: 1300 / 2000 [ 65%]  (Sampling)
## Chain 4 Iteration: 1400 / 2000 [ 70%]  (Sampling)
## Chain 4 Iteration: 1500 / 2000 [ 75%]  (Sampling)
## Chain 4 Iteration: 1600 / 2000 [ 80%]  (Sampling)
## Chain 4 Iteration: 1700 / 2000 [ 85%]  (Sampling)
## Chain 4 Iteration: 1800 / 2000 [ 90%]  (Sampling)
## Chain 4 Iteration: 1900 / 2000 [ 95%]  (Sampling)
## Chain 4 Iteration: 2000 / 2000 [100%]  (Sampling)
## Chain 4 finished in 0.0 seconds.
##
## All 4 chains finished successfully.
## Mean chain execution time: 0.0 seconds.
## Total execution time: 1.1 seconds.
```

```
## Warning: 697 of 4000 (17.0%) transitions ended with a divergence.
## See https://mc-stan.org/misc/warnings for details.
```

```
expo_fit
```

```
##  variable   mean median   sd  mad     q5    q95 rhat ess_bulk ess_tail
##      lp__ -31.94 -31.76 0.53 0.23 -32.91 -31.59 1.00     1632     1491
##      beta   0.16   0.16 0.04 0.04   0.11   0.24 1.00     1351     1491
```

Some people got confused about what was supposed to be where. I am willing to give credit for this part if the above code is somewhere reasonably close to here, on the basis that you are doing about the right things in the right order.

In the second line of code, the thing you sampled from has to be the thing you compiled, in my case `expo` requires the second line to contain `expo$sample`. If you used a different name in the previous part, use *your* name here as well (I will check). Of course, the `data =` has to use the thing you created on the first line. If you called the data in your Stan code `y` rather than `x`, you should also call it `y` here for consistency.

Extra: There are more warnings than we would like to see. The reason is that a uniform prior like this is rather an odd one; we have taken the attitude that β is "almost certainly" between 0.1 and 0.5 and made a prior that has β *certainly* between these two values (never mind about the "almost"). When a value of β (like 0.9) is impossible in the prior, it is actually impossible in the posterior as well, regardless of what the likelihood says. The consequence is that sometimes the sampler will try to sample one of the impossible values. This is where most of the messages are coming from. In practice, you'd use a prior distribution that includes all the values that

β could possibly take (such as a gamma distribution), but I wanted to make the coding part easier for you, and didn't want to have to get into the gamma distribution as well. I figure you probably remember the uniform distribution.

(f) [2] A summary of the simulated posterior distribution of β is shown in Figure 34. What is a 90% posterior interval for β?

My answer:

Pick the `q5` and `q95` values off Figure 34 in the `beta` (second) row: 0.11 to 0.24. Just that.

Extra: this is a bit narrower than the prior distribution (from 0.1 to 0.5), which is usually the case, because even 10 observations will tell us *something* about the parameter we are trying to estimate. In this case, the observations are mostly fairly large, which suggests that β should be fairly small, since the mean is $1/\beta$.

Disclosure: the values in `w` were actually drawn from an exponential distribution with $\beta = 0.2$, which as you see actually *is* in the 90% posterior interval.

(g) [3] Somebody says to you that you have to interpret the interval of the previous part in this way: "in 90% of all possible samples, the procedure will give you an interval that contains the true value of β". Is that what your interval of the previous part says, or not? If not, what does that interval actually say? Explain briefly.

My answer:

The explanation you were given is of a (frequentist, repeated sampling) confidence interval, where there is a "true" value of β that is fixed but unknown to you. But this is no longer the logic we are working under: in Bayesian estimation, the data is given and everything else is random, with distributions. This is why we are trying to find posterior distributions, including the one here (of β). Another way to get to this is that the data in a Bayesian context really is "given", and so there is no conception of what might happen if you had different data.

Because β has a distribution, the interpretation of this interval really is as simple as "β has probability 0.90 of being between 0.11 and 0.24".

That is to say, the interpretation really is what you originally wanted the interpretation of a confidence interval to be; in that case, though, you were still in the repeated-sampling world, and so the interpretation had to be that business about "all possible samples".

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

Figures

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

Figure 1: Packages

```
cultivar_name,alcohol,malic_acid,ash,mg
grignolino,12.87,4.61,2.48,86
grignolino,13.4,4.6,2.86,112
barolo,13.9,1.68,2.12,101
barbera,11.84,0.89,2.58,94
barolo,13.51,1.8,2.65,110
barolo,13.05,1.73,2.04,92
barolo,14.38,1.87,2.38,102
barolo,13.5,1.81,2.61,96
barbera,11.87,4.31,2.39,82
barbera,12,0.92,2,86
barbera,12.21,1.19,1.75,151
barbera,12.52,2.43,2.17,88
```

Figure 2: Wine data (some)

```
wine
```

```
## # A tibble: 178 x 5
##    cultivar_name alcohol malic_acid   ash    mg
##    <chr>           <dbl>      <dbl> <dbl> <dbl>
##  1 grignolino       12.9       4.61  2.48    86
##  2 grignolino       13.4       4.6   2.86   112
##  3 barolo           13.9       1.68  2.12   101
##  4 barbera          11.8       0.89  2.58    94
##  5 barolo           13.5       1.8   2.65   110
##  6 barolo           13.0       1.73  2.04    92
##  7 barolo           14.4       1.87  2.38   102
##  8 barolo           13.5       1.81  2.61    96
##  9 barbera          11.9       4.31  2.39    82
## 10 barbera          12         0.92  2       86
## # i 168 more rows
```

Figure 3: Wine data after being read in (some)

```
##
##      One-sample t test power calculation
##
##               n = 25.38969
##           delta = 10
##              sd = 20
##       sig.level = 0.05
##           power = 0.677
##     alternative = two.sided
```

Figure 4: Power analysis

```
wine %>% group_by(cultivar_name) %>%
  summarize(n = n())
```

```
## # A tibble: 3 x 2
##   cultivar_name     n
##   <chr>         <int>
## 1 barbera          71
## 2 barolo           59
## 3 grignolino       48
```

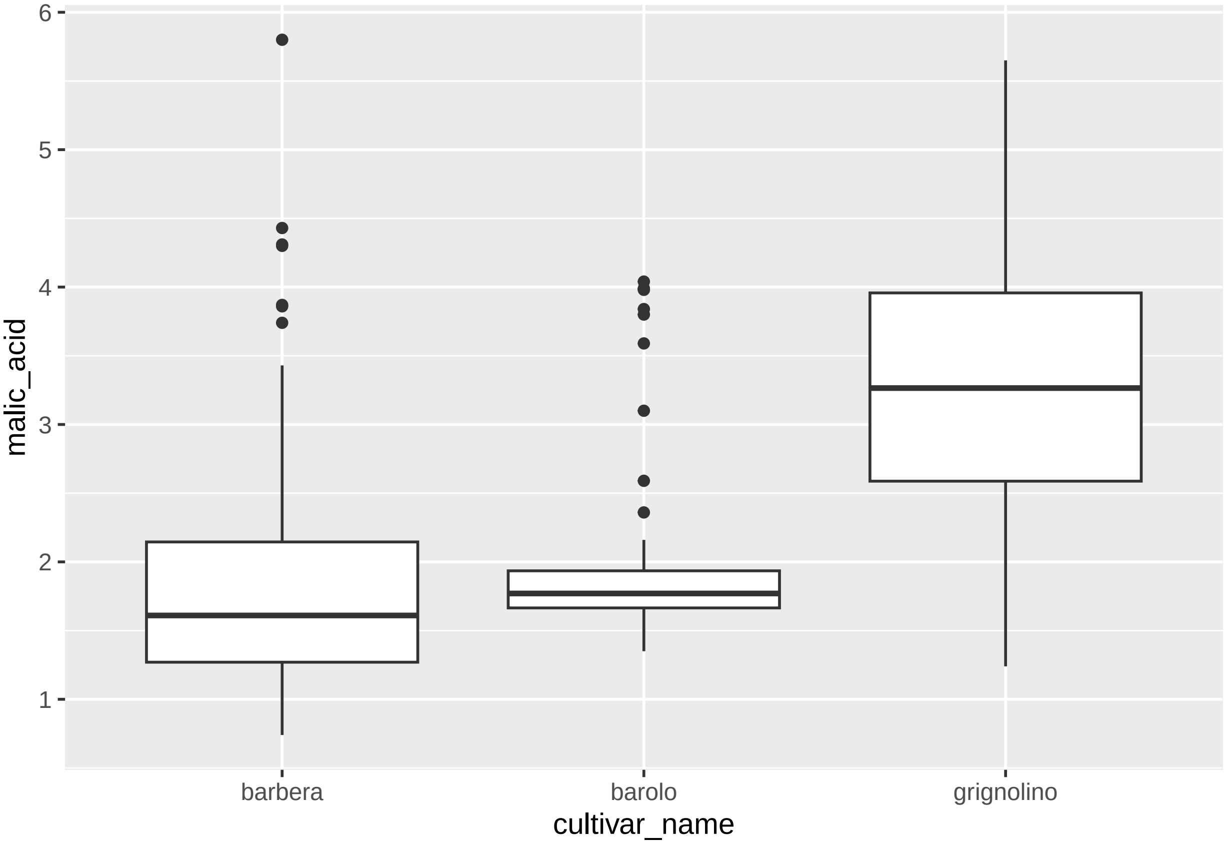


Figure 5: Wine data summary and plot

```
median_test(wine, malic_acid, cultivar_name)
```

```
## $table
##             above
## group        above below
##   barbera       25    46
##   barolo        21    38
##   grignolino    43     5
##
## $test
##        what        value
## 1 statistic 4.119291e+01
## 2        df 2.000000e+00
## 3   P-value 1.135205e-09
```

Figure 6: Wine data Mood Median Test

```
pairwise_median_test(wine, malic_acid, cultivar_name)
```

```
## # A tibble: 3 x 4
##   g1      g2          p_value adj_p_value
##   <chr>   <chr>         <dbl>       <dbl>
## 1 barbera barolo     3.97e- 2    1.19e- 1
## 2 barbera grignolino 1.52e-11    4.55e-11
## 3 barolo  grignolino 2.15e-12    6.44e-12
```

Figure 7: Wine data pairwise median tests

```
wine %>% filter(cultivar_name == "barolo") -> barolo
tibble(sim = 1:10000) %>%
  rowwise() %>%
  mutate(my_sample = list(sample(barolo$malic_acid, replace = TRUE))) %>%
  mutate(my_mean = mean(my_sample)) %>%
  ggplot(aes(sample = my_mean)) + stat_qq() + stat_qq_line()
```

Figure 8: Wine data mystery code

```
d1
```

```
## # A tibble: 2 x 3
##      id     a     b
##   <dbl> <dbl> <dbl>
## 1     1    10    11
## 2     2     8     9
```

Figure 9: Dataframe `d1`

```
d1 %>% pivot_longer(-id, names_to = "name", values_to = "value")
```

Figure 10: Code for dataframe `d1`

```
d2
```

```
## # A tibble: 2 x 5
##     row  m_ht  f_ht  m_wt  f_wt
##   <dbl> <dbl> <dbl> <dbl> <dbl>
## 1     7   180   150    80    60
## 2     8   185   160    90    55
```

Figure 11: Dataframe d2

```
## # A tibble: 8 x 4
##     row gender what  measure
##   <dbl> <chr>  <chr>   <dbl>
## 1     7 m      ht        180
## 2     7 f      ht        150
## 3     7 m      wt         80
## 4     7 f      wt         60
## 5     8 m      ht        185
## 6     8 f      ht        160
## 7     8 m      wt         90
## 8     8 f      wt         55
```

Figure 12: Dataframe d2 output

```
d3
```

```
## # A tibble: 2 x 3
##      id   x_1   x_2
##   <dbl> <dbl> <dbl>
## 1     4    10    11
## 2     6     8     9
```

Figure 13: Dataframe d3

```
d3 %>% pivot_longer(-id, names_to = c(".value", "col"), names_sep = "_")
```

Figure 14: Code for dataframe d3

```
d4
```

```
## # A tibble: 4 x 3
##     row x     measure
##   <dbl> <chr>   <dbl>
## 1     7 m_ht      180
## 2     7 f_ht      150
## 3     7 m_wt       80
## 4     7 f_wt       60
```

Figure 15: Dataframe d4

```
d4 %>% separate(x, into = c("gender", "what"), sep = "_")
```

Figure 16: Code for dataframe `d4`

```
d5
```

```
## # A tibble: 4 x 3
##     row group     x
##   <dbl> <chr> <dbl>
## 1     1 a        14
## 2     1 b        15
## 3     2 a        16
## 4     2 b        17
```

Figure 17: Dataframe `d5`

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

Figure 18: Dataframe `d5` output

```
d6
```

```
## # A tibble: 4 x 3
##   x         y z
##   <chr> <dbl> <chr>
## 1 c        16 low
## 2 b        18 high
## 3 a        20 medium
## 4 b        22 low
```

Figure 19: Dataframe `d6`

```
d6 %>% pivot_wider(names_from = z, values_from = y)
```

Figure 20: Code for dataframe `d6`

```
engel
```

```
## # A tibble: 234 x 2
##    income foodexp
##     <dbl>   <dbl>
##  1   420.    256.
##  2   541.    311.
##  3   901.    486.
##  4   639.    403.
##  5   751.    496.
##  6   946.    634.
##  7   829.    631.
##  8   979.    700.
##  9  1310.    831.
## 10  1492.    815.
## # i 224 more rows
```

Figure 21: Food expenditure data (some)

```
engel.1 <- lm(foodexp ~ income, data  = engel)
summary(engel.1)
```

```
##
## Call:
## lm(formula = foodexp ~ income, data = engel)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -622.00  -54.02    3.22   52.87  398.72
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) 91.33302   15.52094   5.885 1.39e-08 ***
## income       0.54654    0.01458  37.497  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 100.2 on 232 degrees of freedom
## Multiple R-squared:  0.8584, Adjusted R-squared:  0.8578
## F-statistic:  1406 on 1 and 232 DF,  p-value: < 2.2e-16
```

Figure 22: Food expenditure: regression analysis

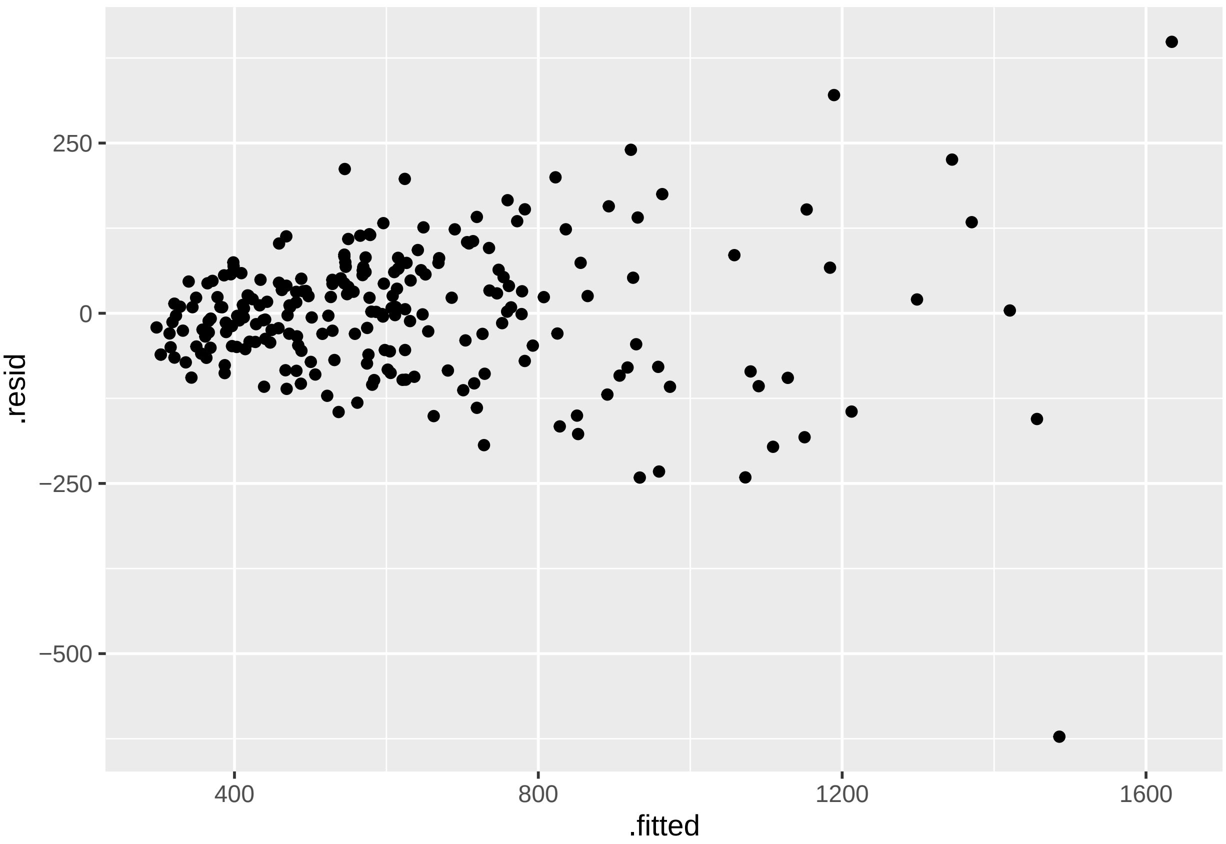


Figure 23: Food expenditure: residual plot 1

```
boxcox(foodexp ~ income, data = engel)
```

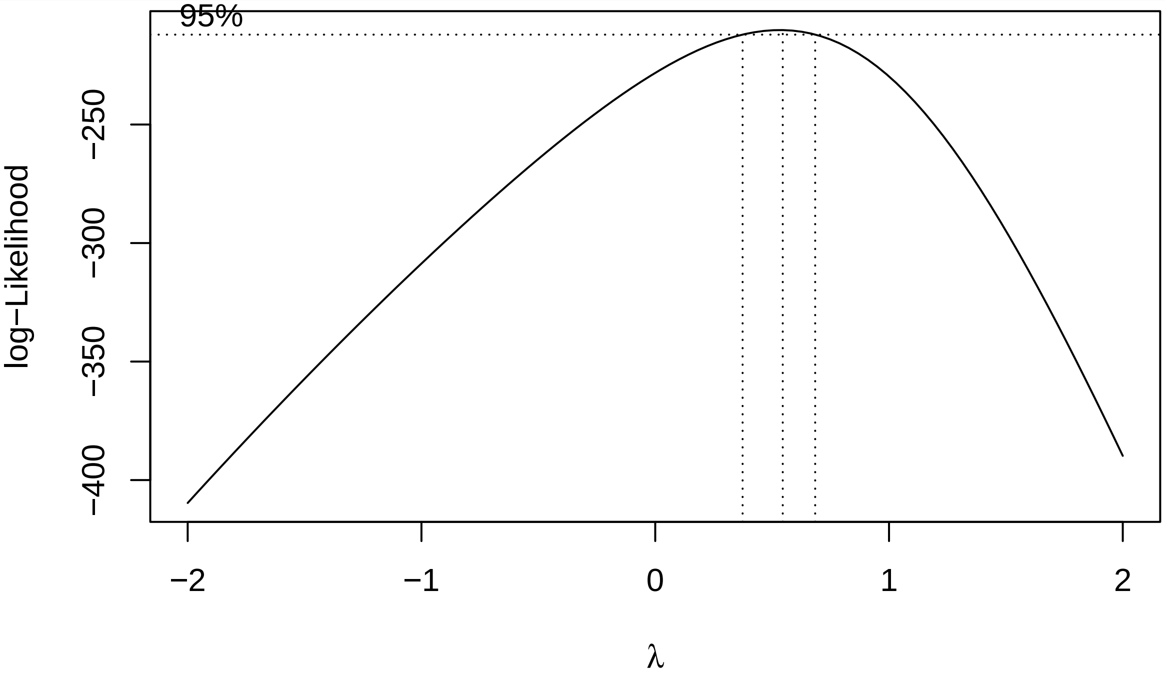


Figure 24: Food expenditure: Box-Cox

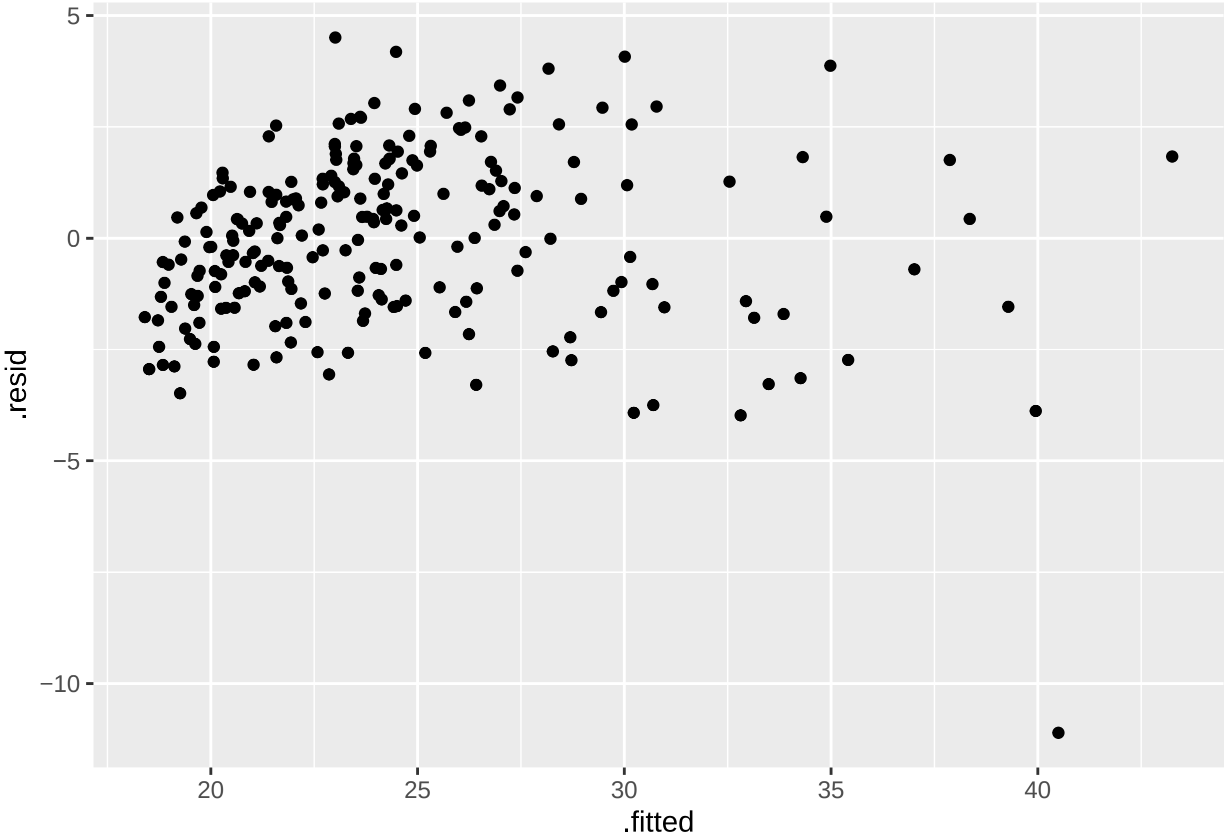


Figure 25: Food expenditure: residual plot 2

- `incomegp` income group (1=lowest, 5=highest)
- `house` security of housing tenure (1=rent, 2=mortgage, 3=owned outright)
- `children` number of children in household
- `singpar` is the respondent a single parent?
- `agegp` age group (1=youngest)
- `bankacc` does the respondent have a bank account?
- `bsocacc` does the respondent have a building society (credit union) account?
- `manage` self-rating of money management skill (high values=high skill)
- `ccarduse` how often did s/he use credit cards (1=never... 3=regularly)
- `cigbuy` does s/he buy cigarettes?
- `xmasbuy` does s/he buy Christmas presents for children?
- `locintrn` score on a locus of control scale (high values=internal)
- `prodebt` score on a scale of attitudes to debt (high values=favourable to debt (response variable)

Figure 26: Debt survey items

```
debt
```

```
## # A tibble: 304 x 13
##    incomegp house children singpar agegp bankacc bsocacc manage ccarduse cigbuy
##       <dbl> <dbl>    <dbl>   <dbl> <dbl>   <dbl>   <dbl>  <dbl>    <dbl>  <dbl>
##  1        3     3        0       0     4       1       0      5        2      0
##  2        5     2        2       0     2       1       0      5        3      0
##  3        3     3        0       0     4       1       0      4        2      0
##  4        4     2        0       0     2       1       0      5        2      0
##  5        4     2        0       0     2       1       0      4        2      0
##  6        2     1        1       0     4       1       0      4        1      0
##  7        2     3        0       0     4       1       0      5        1      0
##  8        2     3        0       0     4       1       0      5        1      0
##  9        2     3        2       0     4       0       1      4        2      0
## 10        2     2        2       1     3       1       0      4        1      1
## # i 294 more rows
## # i 3 more variables: xmasbuy <dbl>, locintrn <dbl>, prodebt <dbl>
```

Figure 27: Debt data (some)

```
debt.1 <- lm(prodebt ~ ., data = debt)
summary(debt.1)
```

```
##
## Call:
## lm(formula = prodebt ~ ., data = debt)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -1.95085 -0.46986 -0.01442  0.40263  1.87677
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  4.04642    0.31682  12.772  < 2e-16 ***
## incomegp     0.06463    0.03373   1.916 0.056336 .
## house       -0.05331    0.06751  -0.790 0.430378
## children     0.03813    0.03898   0.978 0.328749
## singpar      0.02054    0.17372   0.118 0.905984
## agegp       -0.10206    0.04761  -2.144 0.032899 *
## bankacc      0.06248    0.12123   0.515 0.606641
## bsocacc     -0.11198    0.08344  -1.342 0.180628
## manage      -0.12820    0.04556  -2.814 0.005231 **
## ccarduse     0.18779    0.05258   3.571 0.000415 ***
## cigbuy      -0.15448    0.08731  -1.769 0.077894 .
## xmasbuy      0.20147    0.11928   1.689 0.092298 .
## locintrn    -0.13942    0.04371  -3.190 0.001579 **
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.6562 on 291 degrees of freedom
## Multiple R-squared:  0.2043, Adjusted R-squared:  0.1715
## F-statistic: 6.226 on 12 and 291 DF,  p-value: 8.916e-10
```

Using a dot on the right side of a model formula means “all the other variables”.

Figure 28: Debt data regression 1

```
debt.2 <- update(debt.1, .~. - singpar - bankacc - house - children - bsocacc)
summary(debt.2)
```

```
##
## Call:
## lm(formula = prodebt ~ incomegp + agegp + manage + ccarduse +
##     cigbuy + xmasbuy + locintrn, data = debt)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -1.99736 -0.43552  0.00559  0.40031  1.81132
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  4.08091    0.29233  13.960  < 2e-16 ***
## incomegp     0.06025    0.03063   1.967 0.050125 .
## agegp       -0.13047    0.04143  -3.149 0.001805 **
## manage      -0.14141    0.04389  -3.222 0.001416 **
## ccarduse     0.18775    0.05149   3.647 0.000314 ***
## cigbuy      -0.13220    0.08560  -1.544 0.123579
## xmasbuy      0.22305    0.11479   1.943 0.052963 .
## locintrn    -0.14165    0.04330  -3.271 0.001198 **
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.6554 on 296 degrees of freedom
## Multiple R-squared:  0.1926, Adjusted R-squared:  0.1735
## F-statistic: 10.09 on 7 and 296 DF,  p-value: 2.546e-11
```

`update` requires a model to update, and then how to update it. This one means "leave everything the same except take out the five explanatory variables listed."

Figure 29: Debt data regression 2

```
anova(debt.2, debt.1)
```

```
## Analysis of Variance Table
##
## Model 1: prodebt ~ incomegp + agegp + manage + ccarduse + cigbuy + xmasbuy +
##     locintrn
## Model 2: prodebt ~ incomegp + house + children + singpar + agegp + bankacc +
##     bsocacc + manage + ccarduse + cigbuy + xmasbuy + locintrn
##   Res.Df    RSS Df Sum of Sq      F Pr(>F)
## 1    296 127.14
## 2    291 125.31  5     1.836 0.8528 0.5134
```

Figure 30: Debt data: a test

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

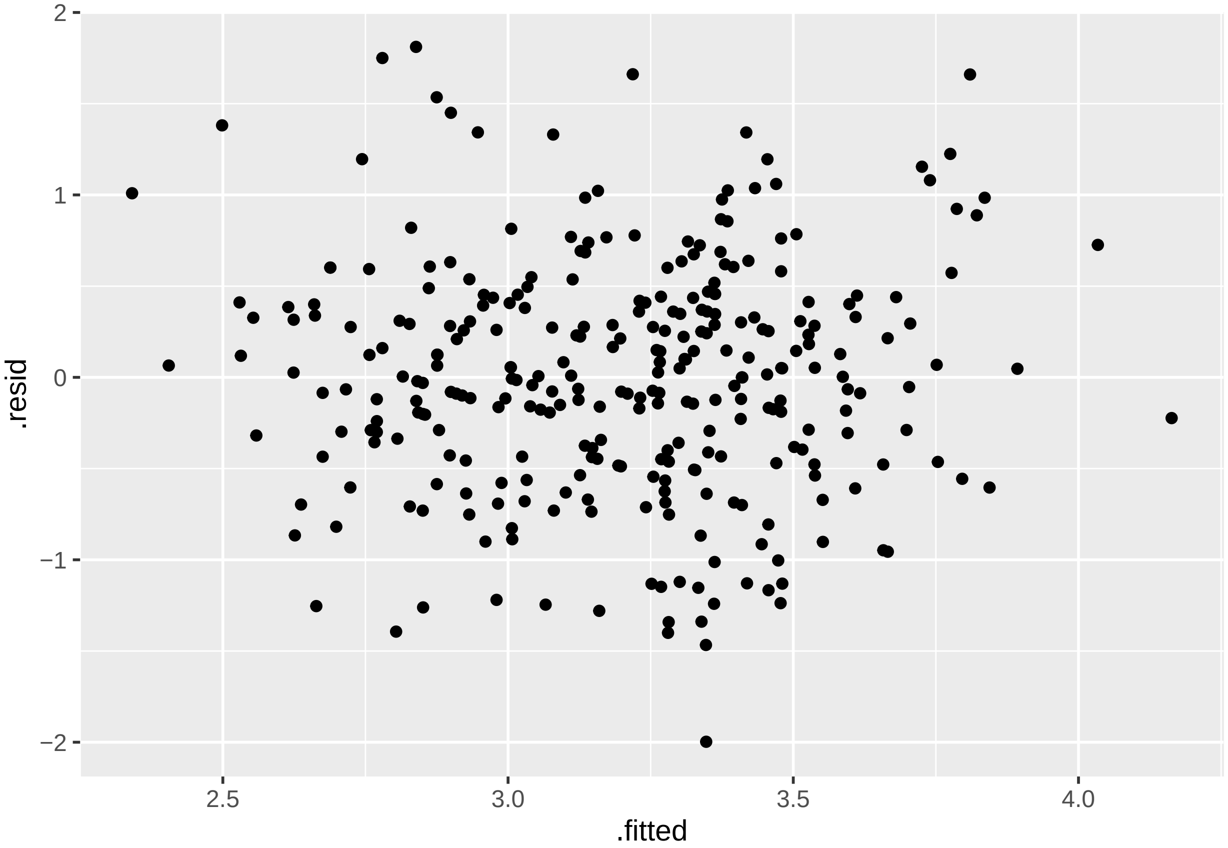


Figure 31: Debt data: residuals vs. fitted values from model `debt.2`

```
ggplot(debt.2, aes(sample = .resid)) +
  stat_qq() + stat_qq_line()
```

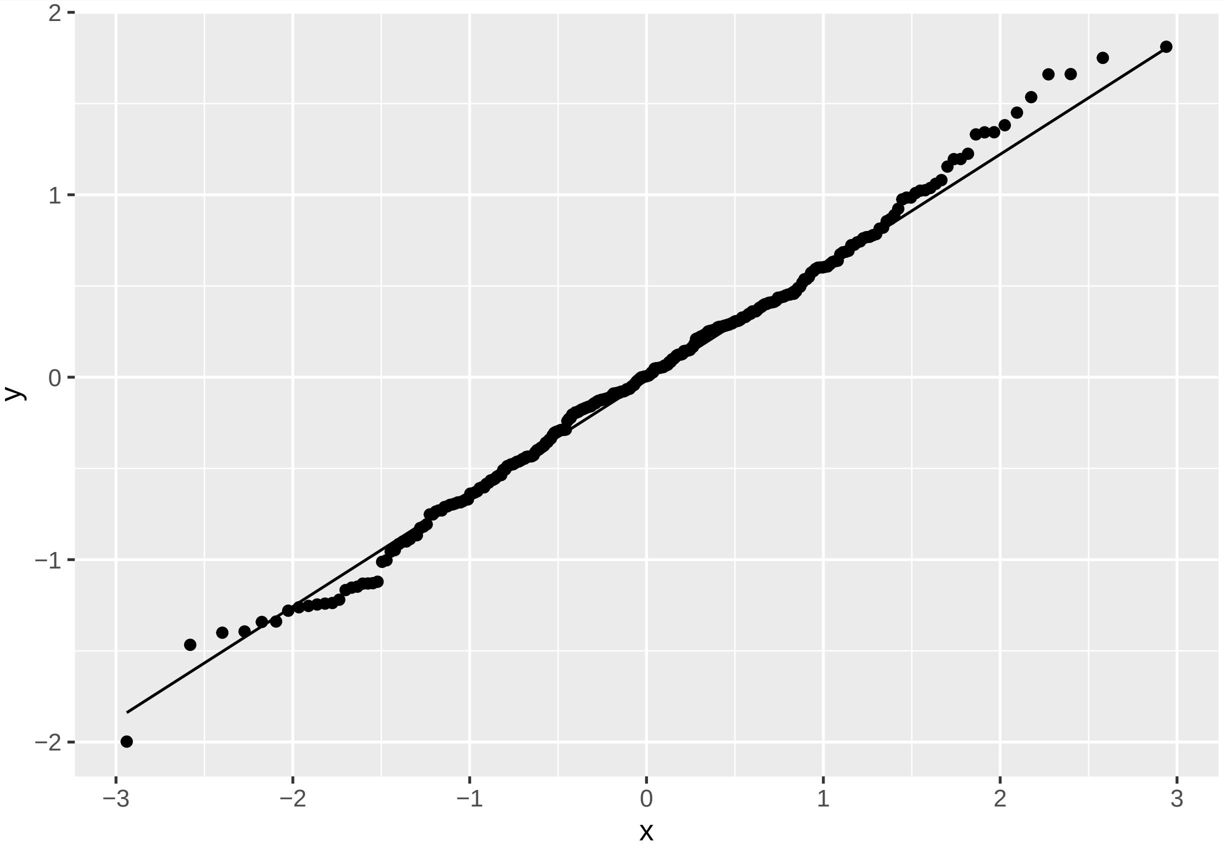


Figure 32: Debt data: normal quantile plot of residuals from model `debt.2`

```
w <- c(0.5, 5.4, 3.7, 13.8, 12.9, 4.0, 17.3, 6.6, 4.8, 2.5)
```

Figure 33: Observed data for estimating β by Bayesian methods

```
expo_fit
```

```
##  variable   mean median   sd  mad     q5    q95 rhat ess_bulk ess_tail
##      lp__ -31.94 -31.76 0.53 0.22 -32.80 -31.59 1.00     1379     1212
##      beta   0.16   0.16 0.04 0.04   0.11   0.24 1.00     1004     1154
```

Figure 34: Summary of posterior distribution of β