Assignment 3

Due Thursday September 26 at 11:59pm on Quercus

As before, the questions without solutions (here, the last two) are an assignment: you need to do these questions yourself and hand them in (instructions below).

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

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

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

As ever, you'll want to begin with:

```
library(tidyverse)
```

1. Work through problems 7.1 through 7.4 in Chapter 7 of PASIAS. This will prepare you for the fertilizer question below.

2. Work through problem 8.1 of PASIAS. This will prepare you for the question about the exponential distribution below.

3. Corn plants are treated with one of two fertilizers, called A and B. The amount of corn produced by each plant is measured; this is called the "yield". We are interested in seeing whether the mean yield differs between the two fertilizers. The available plants (which you can think of as a sample of "all possible plants") were randomly assigned to fertilizers. The data are in `http://www.utsc.utoronto.ca/~butler/assgt_data/ferto.txt`, the values separated by a single space.

   (a) (3 marks) Read in and display (at least some of) the data. How many plants got each fertilizer? (You should get R to figure this out rather than counting them by hand.)

   Solution:

   "Separated by a single space" means `read_delim`, so:

```
my_url <- "http://www.utsc.utoronto.ca/~butler/assgt_data/ferto.txt"
ferto <- read_delim(my_url, " ")
```

```
## Parsed with column specification:
## cols(
##   fertilizer = col_character(),
##   yield = col_double()
## )
```

```
ferto
```

```
## # A tibble: 29 x 2
##    fertilizer yield
##    <chr>      <dbl>
##  1 A            452
##  2 A            874
##  3 A            554
##  4 A            447
##  5 A            356
##  6 A            754
##  7 A            558
##  8 A            574
##  9 A            664
## 10 A            682
## # ... with 19 more rows
```

This is enough for "some of" the data, though for yourself you should probably scroll down and check that you have representatives from both fertilizers.

I gave my data frame the same name as the file, since one of the columns in it is called `fertilizer` and I didn't want to confuse things.

To count how many of each fertilizer, `count`, thus:

```
ferto %>% count(fertilizer)
```

```
## # A tibble: 2 x 2
##   fertilizer     n
##   <chr>      <int>
## 1 A             13
## 2 B             16
```

13 plants with fertilizer A and 16 with fertilizer B. This is all right for a two-sample test (coming up), since the groups don't have to be the same size.

(b) (3 marks) Run a two-sample t-test to compare the mean yields of plants given the two different fertilizers. What do you conclude, in the context of the data?

Solution:

With the data in long format (see the Extra to the previous part), this is exactly what you would expect. We are looking for any difference (implied by "compare"), so the test is the default two-sided:

```r
t.test(yield~fertilizer, data=ferto)
```

```
##
##  Welch Two Sample t-test
##
## data:  yield by fertilizer
## t = -0.15135, df = 19.169, p-value = 0.8813
## alternative hypothesis: true difference in means is not equal to 0
## 95 percent confidence interval:
##  -120.2734  104.0426
## sample estimates:
## mean in group A mean in group B
##        549.3846        557.5000
```

The null hypothesis is that the (population) mean yields for plants given the two fertilizers are the same; the alternative is that the means are different.

The P-value is 0.8813, much bigger than 0.05, so we do not reject the null hypothesis; there is no evidence that the mean corn yields differ between the fertilizers. (They might actually be the same or they might not, but we couldn't prove that they were different.)

Don't try to "reject the alternative" here. That's not how it works.

(c) (3 marks) Obtain the confidence interval from your output of the previous part. What precisely is this a confidence interval for? How is it consistent with the result of your test? Explain briefly.

Solution: The confidence interval is −120.3 to 104.0. (Remember to round off to some sensible number of decimals.) This is a confidence interval for the *difference* in (population) mean corn yields between fertilizer A and fertilizer B. This is actually A minus B; you can tell because A is before B alphabetically, and also right at the bottom the mean for fertilizer A is given first, before the one for fertilizer B.

This is consistent with the test (in which we couldn't reject the null that the means were the same) because a difference in means of zero is inside the confidence interval; if you like, zero is a "plausible" value for the difference in means, which says that (i) you wouldn't reject it in a test, and (ii) it would be inside a confidence interval.

(d) (2 marks) Do you think a 99% confidence interval would contain zero for these data? Explain briefly why or why not.

Solution: There are a couple of ways to argue this.

The easier one is to say that a 99% confidence interval would be wider (longer) than a 95% interval, so would include all the values of your interval above and more. The 95% CI contains zero, so the 99% one must contain zero as well.

A more sophisticated argument is the one used in "duality of test and CI" later. It's easiest to think of this one in terms of "plausible": at the 95% confidence level, zero is plausible as a difference of means and is also not rejected because the P-value is greater than 0.05. But the P-value is also (much) greater than 0.01, so a difference in means of zero would not be rejected at the 0.01 level either, and therefore zero would be inside the corresponding, 99% confidence interval as well.

Of course, there's nothing stopping you doing this:

```
t.test(yield~fertilizer, data=ferto, conf.level=0.99)
```

```
##
##  Welch Two Sample t-test
##
## data:  yield by fertilizer
## t = -0.15135, df = 19.169, p-value = 0.8813
## alternative hypothesis: true difference in means is not equal to 0
## 99 percent confidence interval:
##  -161.3665  145.1357
## sample estimates:
## mean in group A mean in group B
##        549.3846        557.5000
```

which will get you the answer, but not why, which you will still have to reason out. (Seeing that this confidence interval is wider than the 95% one might be enough to give you a clue.)

The second argument is more sophisticated because it will also tell you that zero is inside *narrower* confidence intervals, eg. a 60% one, because the P-value is greater than $1-0.60 = 0.40$.

This argument works for two-sided tests as this one is, because confidence intervals are themselves two-sided. You can make it work for a one-sided test as well, but you have to be more careful.

(e) (4 marks) Run a pooled *t*-test for these data, and, by looking at a suitable graph, explain briefly whether you prefer it to the original (Welch) test.

Solution: The key is `var.equal=T`:

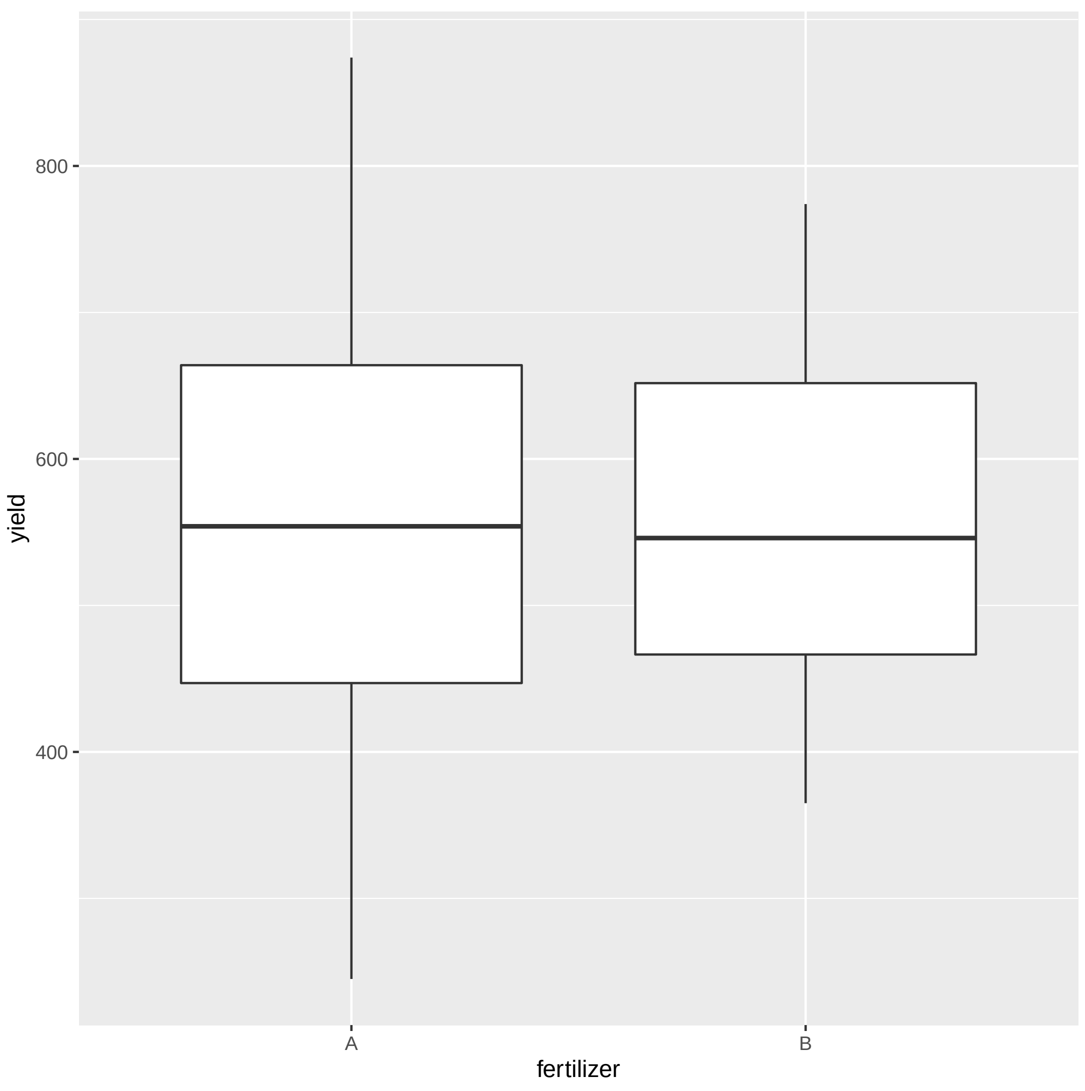
```
t.test(yield~fertilizer, data=ferto, var.equal=T)
```

```
##
##  Two Sample t-test
##
## data:  yield by fertilizer
## t = -0.15877, df = 27, p-value = 0.875
## alternative hypothesis: true difference in means is not equal to 0
## 95 percent confidence interval:
##  -112.99323   96.76246
## sample estimates:
## mean in group A mean in group B
##        549.3846        557.5000
```

You might observe that the P-value is very close to that of the Welch test, 0.875 vs. 0.8813 (but you don't have to). The suggestion from that is that it doesn't matter which flavour of two-sample *t* you do.

To choose between the two tests, make a boxplot, and use it to make a call about whether you think the two groups have sufficiently equal spread or not:

```
ggplot(ferto, aes(x=fertilizer, y=yield)) + geom_boxplot()
```



The spread for fertilizer A is slightly bigger than for fertilizer B. If you think that's enough of a difference to call the spreads different, you should prefer the Welch test; if not, the pooled test is good. Pick one and argue for it; I don't mind which way you go. (Since the P-values are almost identical, it actually doesn't matter which one you choose.)

Another way of arguing for the Welch test is that it is always pretty good, whether you think the spreads are equal or not. If you are unsure which way to go based on your boxplot, that is a reasonable argument for going with Welch.

Extra: the other thing you get from your boxplot is that the medians are almost identical, so it is going to be very difficult to detect a real difference between them (or the means).

(f) (2 marks) The data as I originally received it looked like this:

```
ferto_wide=read_delim("ferto_wide.txt", " ")

## Parsed with column specification:
## cols(
##   A = col_double(),
##   B = col_double()
## )
```

```
ferto_wide
```

```
## # A tibble: 16 x 2
##         A     B
##     <dbl> <dbl>
##  1   452   546
##  2   874   547
##  3   554   774
##  4   447   465
##  5   356   459
##  6   754   665
##  7   558   467
##  8   574   365
##  9   664   589
## 10   682   534
## 11   547   456
## 12   435   651
## 13   245   654
## 14    NA   665
## 15    NA   546
## 16    NA   537
```

Explain briefly how this is not an appropriate data layout for us to use to do a two-sample *t*-test with.

Solution:

This is "wide format", with one column per group (fertilizer). This is not as we are accustomed to seeing it, with all the yields in *one* column and a second column containing the groups (which in this case would be the values A and B for the two fertilizers).

That is to say, explain what we need and how we don't have it.

Extra: the `NA` at the bottom are missing values, because more plants got fertilizer B than fertilizer A, but the columns of a data frame have to be all the same length, and therefore the shorter one gets filled in with missings.

It does actually work for a two-sample *t*-test; what you do is to give the two column names separately to `t.test`. This one doesn't accept a `data`, so you have to use `with` (or dollar signs):

```
with(ferto_wide, t.test(A, B))
```

```
##
##  Welch Two Sample t-test
##
## data:  A and B
## t = -0.15135, df = 19.169, p-value = 0.8813
## alternative hypothesis: true difference in means is not equal to 0
## 95 percent confidence interval:
##  -120.2734  104.0426
## sample estimates:
## mean of x mean of y
##  549.3846  557.5000
```

but this is not how you've seen it, and I didn't want to confuse the issue. I don't actually like this for a two-sample *t*-test because the layout might make you think that the data are paired when they're actually not. (Plant 1 for fertilizer A and plant 1 for fertilizer B are two *different* plants.) Historically, this is how you originally *had* to use `t.test` to do a two-sample *t*-test; the "formula interface" (the thing with the squiggle) and the `data=` came along later.

Later, we learn about data tidying. This is a common rearrangement that we need to make, and it goes like this:

```
ferto_wide %>%
  gather(fertilizer, yield, everything(), na.rm=T) -> ferto
ferto
```

```
## # A tibble: 29 x 2
##    fertilizer yield
##    <chr>      <dbl>
##  1 A            452
##  2 A            874
##  3 A            554
##  4 A            447
##  5 A            356
##  6 A            754
##  7 A            558
##  8 A            574
##  9 A            664
## 10 A            682
## # ... with 19 more rows
```

Now `ferto` is the way we are accustomed to seeing things, and so I saved it for you to use (this is the actual data you used earlier):

```
write_delim(ferto, "ferto.txt", " ")
```

`write_delim` is the opposite of `read_delim`: instead of getting data from a file and putting it into a data frame, it gets data from a data frame and puts it into a file (that can then be read in with `read_delim`).

4. The exponential is a continuous, positive-valued probability distribution. We will investigate its shape shortly. The R function `rexp` draws random samples from an exponential distribution. It needs two inputs: a sample size, and a parameter called the "rate" that is one over the mean.

(a) (2 marks) Draw a random sample of 100 values from an exponential distribution with mean 10. Save the values in a data frame, and display at least some of the data frame.

Solution:

The first thing to think about is what the “rate” needs to be. One over the mean here is $1/10 = 0.1$.

You have a couple of choices here. The first one that came to my mind is to save the values in a *vector*, and then make a data frame out of this vector:

```
v <- rexp(100, 0.1)
v

##   [1]  0.86362855  1.15442490 12.69113048  5.04784849 15.36762465
##   [6]  5.21655996 11.79344867 15.09842111 11.86569666  0.03228995
##  [11] 10.51774479  0.33904656 10.08248704 25.18791977  2.02700240
##  [16]  5.98190110  6.23217439 17.47448918 19.47733905  2.85998192
##  [21]  6.45193821  2.10172950  0.29584802  2.93365432  6.55140982
##  [26]  8.06402585 20.64366531  9.65830892  2.19900100 26.25494530
##  [31]  5.07567050  3.54497688  4.56574812 22.51601534  0.56876107
##  [36] 16.25551866 22.79218086  1.96021576  7.31997114  2.30725003
##  [41] 19.32305297  4.73647088 27.10524334  4.79756278 26.53679391
##  [46]  8.18027143  7.85120720  2.94908795  2.97077928  7.31867228
##  [51]  5.76585832 42.44414480  3.96342136 18.74287328 16.62996836
##  [56]  5.77084264  8.30381367  7.07102306  4.40282139 15.21353140
##  [61] 11.80475866  2.54876659 12.59312293  3.48665712  7.58249083
##  [66]  7.36754132 10.83976515  1.85496012  0.45912390  9.08849950
##  [71] 51.18776920  0.33893889  0.67367913  5.48363646  8.78060596
##  [76]  0.20731618  3.26497437 12.94068976  0.09985353  1.45631694
##  [81]  2.17611582  0.79628640  4.42036260  1.93071027 12.34911299
##  [86]  8.32652459 22.06573995  3.56115366 13.74095032  0.92890850
##  [91]  8.25791380 10.13804895  1.39069604  0.86635166  7.40128723
##  [96]  2.36149483  5.82677760  6.55516631  7.40415727 18.89300158

d=tibble(v)
d

## # A tibble: 100 x 1
##          v
##      <dbl>
##  1  0.864
##  2  1.15
##  3 12.7
##  4  5.05
##  5 15.4
##  6  5.22
##  7 11.8
##  8 15.1
##  9 11.9
## 10  0.0323
## # ... with 90 more rows
```

I displayed my vector as well (to convince myself that I was creating the right thing), but you don’t have to.

Another way is to create the data frame first off:

```
d1=tibble(v=rexp(100, 0.1))
d1
```

```
## # A tibble: 100 x 1
##        v
##    <dbl>
##  1 22.7
##  2  5.25
##  3  4.82
##  4  3.67
##  5 28.0
##  6  5.76
##  7 57.7
##  8  2.88
##  9 18.9
## 10  1.64
## # ... with 90 more rows
```

Either of these is good. Your values will probably be different from mine, but if you are doing the right thing, it is good.

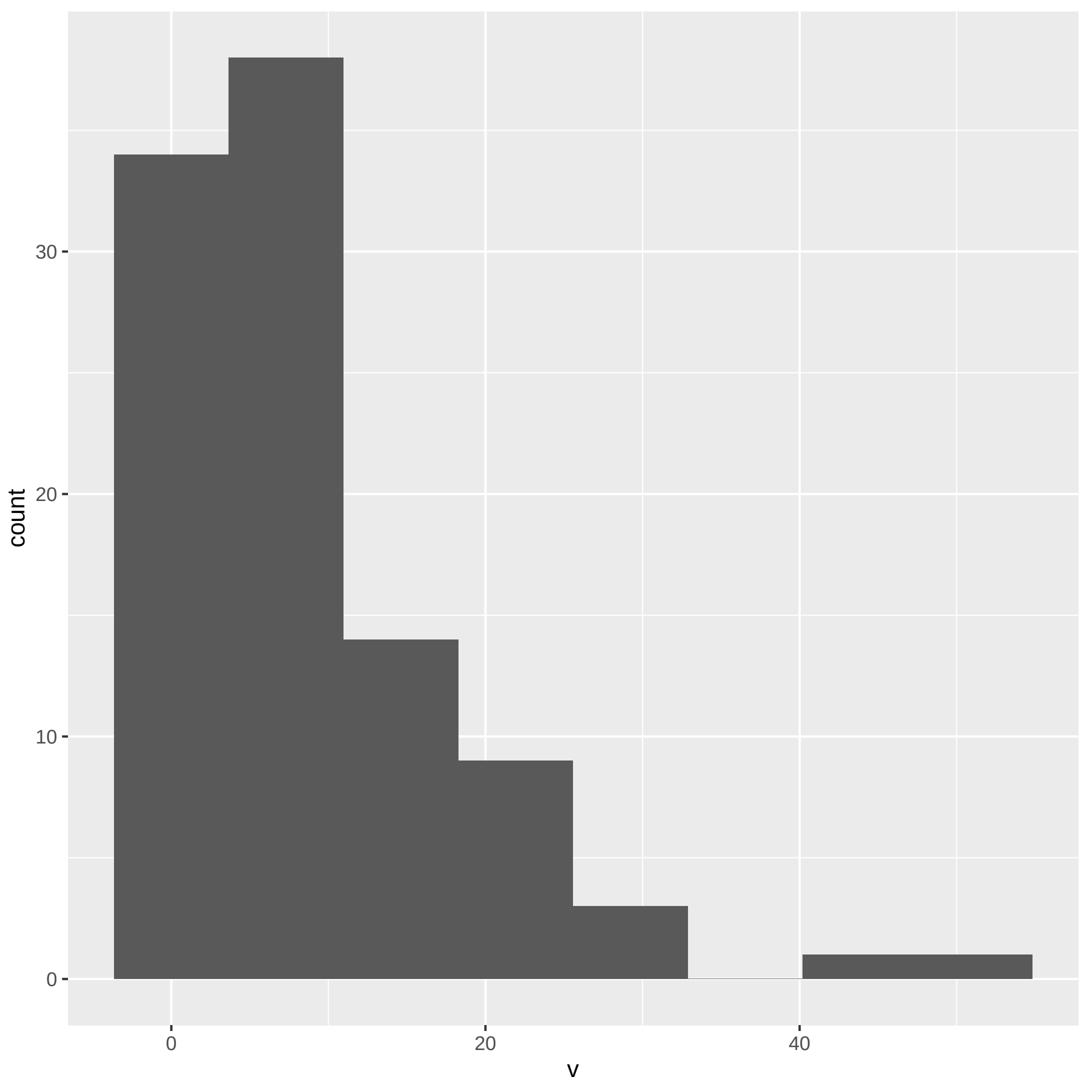
Extra: if you eyeball these, it seems at least somewhat plausible that the mean is 10, though some of the values look a *lot* bigger than 10. Maybe this makes you suspicious about the shape, which would clue you in to the likely answer to the next part.

(b) (3 marks) Make a histogram of your random sample. Why might you have doubts about using a t-test on data from this exponential distribution? Explain briefly.

Solution:

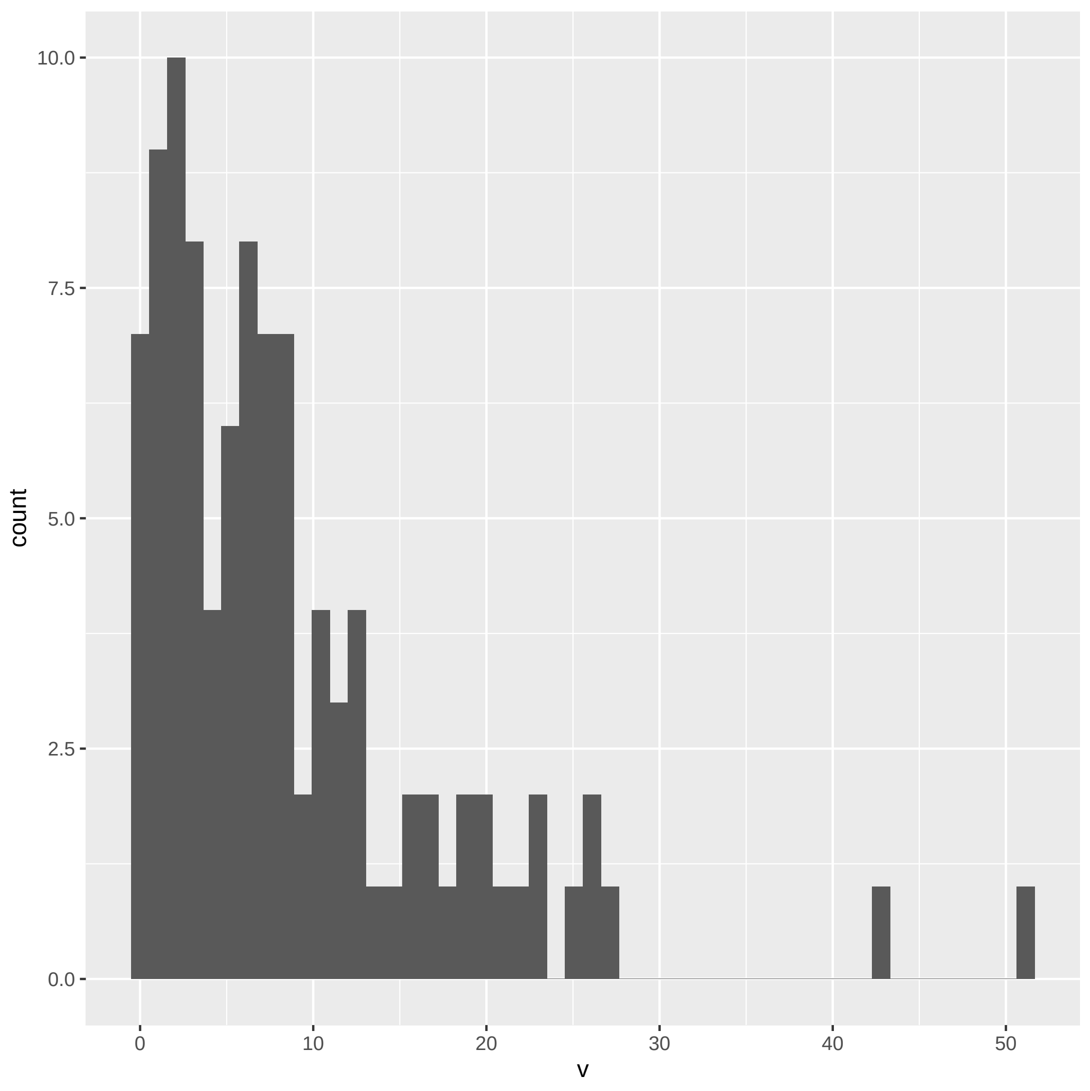
You might guess that something will be up with the shape. With 100 observations, you could justify anything up to about 10 bins:

```
ggplot(d, aes(x=v))+geom_histogram(bins=8)
```



You want enough bins to show that kind of shape clearly, and not so many that the overall picture gets lost in noise:

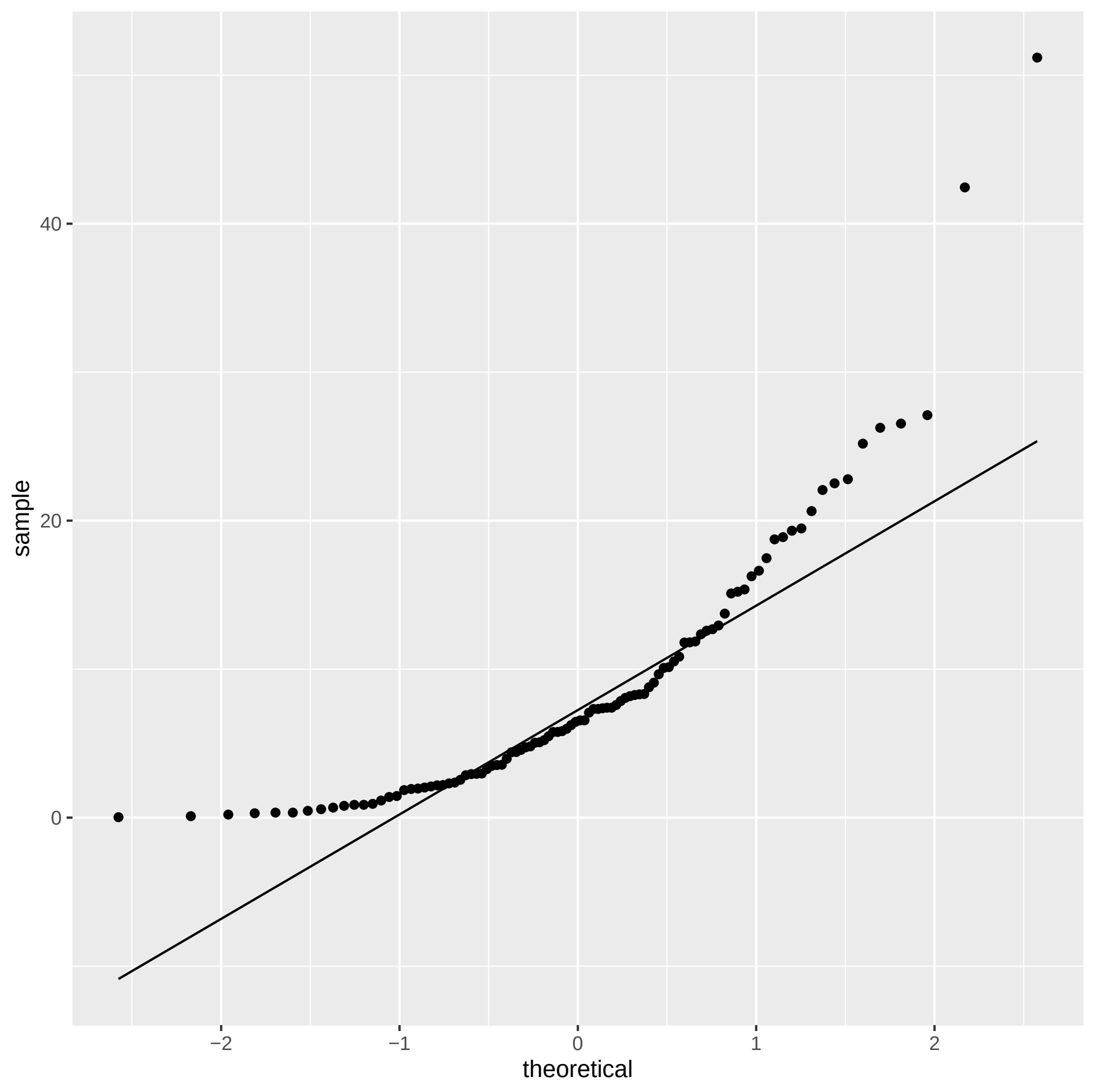
```
ggplot(d, aes(x=v))+geom_histogram(bins=50)
```



This appears to show bimodality (two peaks), which I think is mostly a happenstance of having chosen so many bins. This is mostly driven by the fifth bin happening to have only four observations in it. Those bins around 20 have only one observation or two in them, with detracts from the clarity of the shape.

Another plot you might consider (at least later, after you have seen it, and had I not asked you for a histogram) is a normal quantile plot, since normality is going to be the issue here:

```
ggplot(d, aes(sample=v))+stat_qq()+stat_qq_line()
```



All of these plots are showing strong right skewness. This doesn't look anything like normal, and even with a sample size of 100 (so that the central limit theorem will help quite a bit), you would expect the t-test to be in bad shape.

Again, since your data is probably different from mine, your plots will be different, but the overall impression should be about the same. The evidence for skewness might be in a short left tail, a long right tail, or both.

(c) (4 marks) Generate a large number of random samples of size 100 from an exponential distribution with mean 10. For each one, run a (two-sided) t-test that the mean is 10. For each of those t-tests, extract the P-value. Count up and display how many of those P-values are less than 0.05. (This is the same idea as power by simulation.)

Solution: This is in fact not actually power, as we see in a minute, but you get it the same way. My "large number" of random samples is 10000; you can do any number about 1000 or bigger. (More gives a more accurate answer, but takes longer to get it.)

```
rerun(10000, rexp(100, 0.1)) %>%
  map(~t.test(., mu=10)) %>%
  map_dbl("p.value") -> pvals
tibble(pvals) %>% count(pvals<0.05)
```

```
## # A tibble: 2 x 2
##   `pvals < 0.05`     n
##   <lgl>          <int>
## 1 FALSE           9420
## 2 TRUE             580
```

One point for each of those things.

(d) (3 marks) In the previous part, was the null hypothesis actually true or false? Would rejecting it be correct or some kind of error? Out of your "large number" of random samples, what percentage of them would you expect to result in rejection of the null hypothesis? What percentage actually did? Explain briefly.

Solution: In the previous part, the null hypothesis is actually *true*, since our simulated samples had mean 10, and we were also testing that the population mean was 10 (which it is). Since the null hypothesis is correct, rejecting it would be a type *I* error. Thus, we should expect to incorrectly reject it 5% of the time (we counted how many of the P-values were less than 0.05). I actually did 5.8% of the time. You can say that this is clearly bigger than 5%, or fairly close to 5%. Up to you. (The explanation ought to include some kind of comparison.)

If the null hypothesis had been *wrong*, the above would have been an estimate of the power rather than the probability of a type I error, and then we wouldn't have known what to expect (the actual answer is what we would have been interested in).

Extra: the difference between 5.8% and 5% actually could be for one of *two* reasons: one, that it's a simulation rather than the truth, and two, that the population distribution is actually exponential rather than normal. I did a bigger simulation than usual in an attempt to nail down the first one; my suspicion is that the true type I error rate for this test actually is a little bit bigger than 5%, but it is very close.

(e) (3 marks) Use simulation to estimate the power of the t-test to (correctly) reject a null mean of 13 when the true mean is actually 10 and the data is a sample of size 100 from an exponential distribution with rate 0.1.

Solution: This is exactly the same process as the previous part. It's just a question of what to change. What's different is that the null hypothesis for the t-test is now that the mean is 13. The true mean is still 10, so that part hasn't changed. That leads us to this, copying, pasting and editing:

```
rerun(10000, rexp(100, 0.1)) %>%
  map(~t.test(., mu=13)) %>%
  map_dbl("p.value") -> pvals
tibble(pvals) %>% count(pvals<0.05)
```

```
## # A tibble: 2 x 2
##   `pvals < 0.05`     n
##   <lgl>          <int>
## 1 FALSE           1845
## 2 TRUE            8155
```

The null hypothesis is now wrong, so we want to reject the null hypothesis (get a P-value less than 0.05) and the percentage of times we do, 81.55%, is our estimate of the power.

I think this is nice and high, though the sample size is also large, so maybe I could have guessed this.

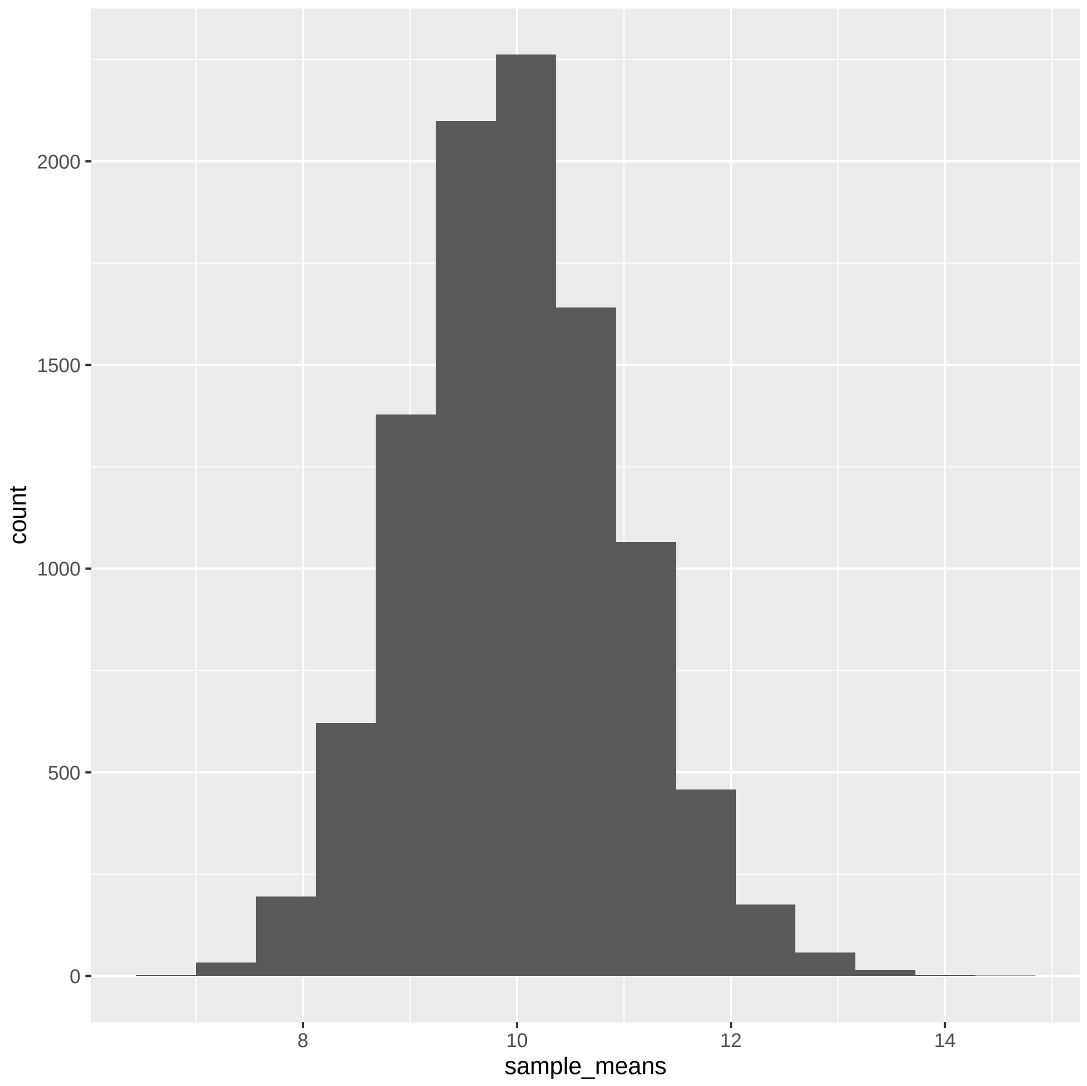
Extra: you might be having your doubts about pursuing the t-test here, given the serious skewness in the exponential distribution. One of the reasons I had you do part (c) was to show you that, despite the skewness, the t-test gives something close to the right answer for testing whether the population mean is indeed 10. This happens because of the large sample; even though the data distribution is not anything like normal, once you get up to a sample size of 100, the Central Limit Theorem is really helping and the *sampling distribution of the sample mean*, which is what really counts, must be pretty close to normal. Therefore, it is actually pretty reasonable to use the t-test at all, and therefore the power estimation is interesting.

An adaptation of the above would simulate the sampling distribution of the sample mean directly:

```
rerun(10000, rexp(100, 0.1)) %>%
  map_dbl(~mean(.)) -> sample_means
```

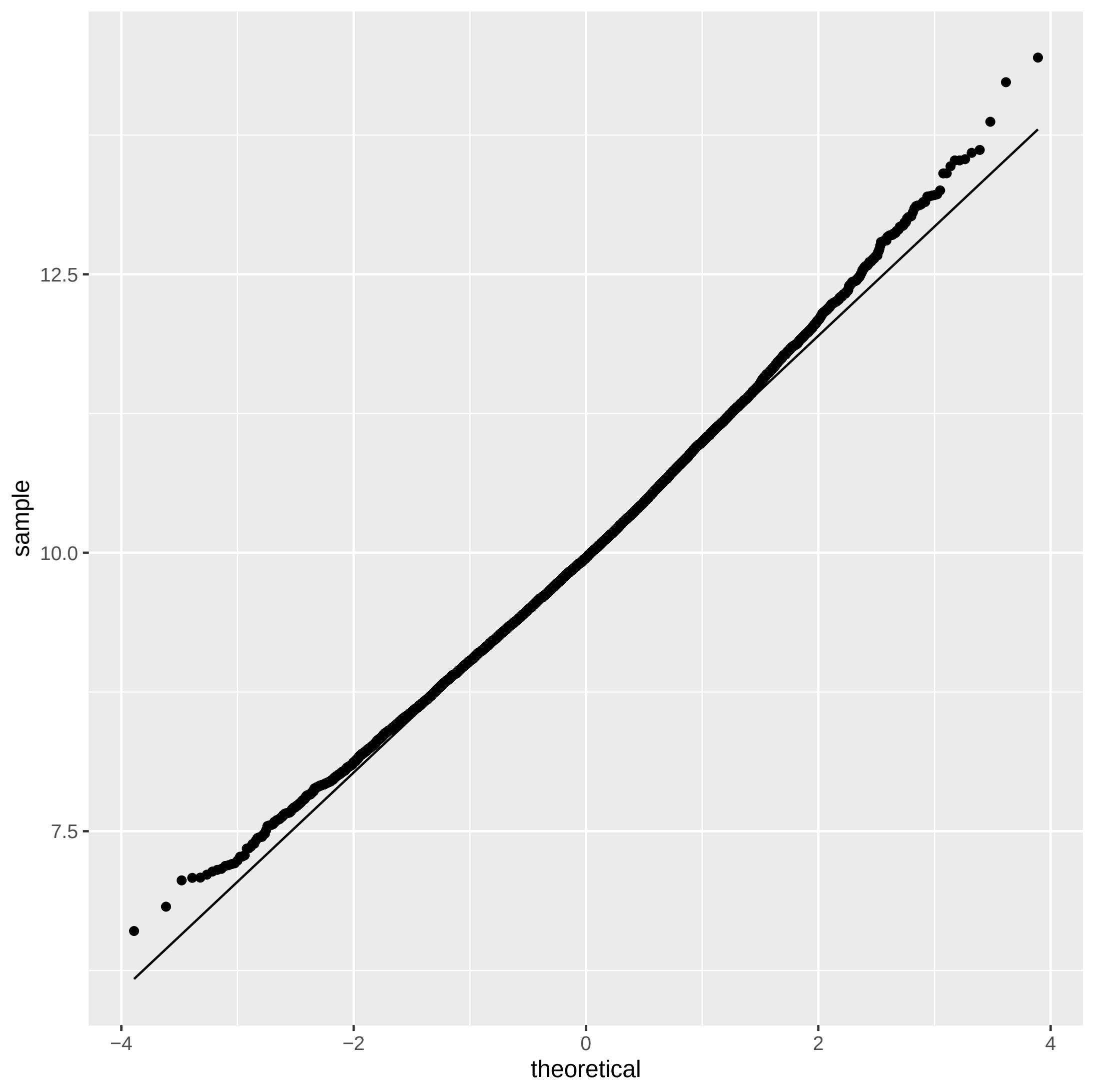
The first line is the same: generate 10000 ("many") random samples of size 100 from an exponential distribution with mean 10 (rate 0.1). The second line says "for each random sample, calculate its mean" and save it in `sample_means`. This is an answer to the question "what kind of sample mean might I get?" So now we take a look at this distribution, either via a histogram:

```
d <- tibble(sample_means)
ggplot(d, aes(x=sample_means)) + geom_histogram(bins=15)
```



or with a normal quantile plot:

```
ggplot(d, aes(sample=sample_means))+stat_qq()+stat_qq_line()
```



That's a little skewed right, but pretty close to being normal, and so we ought to have no real problems with using the *t*-test, because the sample size seems large enough to overcome the skewness.

Extra extra: we had to simulate the power here, because the data distribution wasn't normal (which is what `power.t.test` assumes).