Assignment 3

Instructions: Make an R Notebook and in it answer the questions below. When you are done, hand in on Quercus the *output* from Previewing (or Knitting) your Notebook. Do *not* hand in the Notebook itself. You want to show that you can (i) write code that will answer the questions, (ii) run that code and get some sensible output, (iii) write some words that show you know what is going on and that reflect your conclusions about the data. Your goal is to convince the grader that you *understand* what you are doing: not only doing the right thing, but making it clear that you know *why* it's the right thing.

Do *not* expect to get help on this assignment. The purpose of the assignments is for you to see how much *you* have understood. You will find that you also learn something from grappling with the assignments. The time to get help is after you watch the lectures and work through the problems from PASIAS, via tutorial and the discussion board, that is *before* you start work on the assignment. The only reasons to contact the instructor while working on an assignment are to report (i) something missing like a data file that cannot possibly be read, (ii) something *beyond your control* that makes it impossible to finish the assignment in time after you have started it.

There is a time limit on this assignment (you will see Quercus counting down the time remaining).

1. A population of a large number of values `v` is at http://ritsokiguess.site/STAC32/pop.csv, in a CSV file.

   (a) Read in the population and display some of the values.

Solution:

```
my_url <- "http://ritsokiguess.site/STAC32/pop.csv"
pop <- read_csv(my_url)
```

```
##
## -- Column specification --------------------------------------------------------
## cols(
##   v = col_double()
## )
```

```
pop
```

```
## # A tibble: 10,000 x 1
##        v
##    <dbl>
##  1  9.97
##  2  2.18
##  3  6.20
##  4  2.11
##  5  6.30
##  6  1.54
##  7  5.77
##  8  2.94
##  9 16.8
## 10  1.95
## # ... with 9,990 more rows
```

10,000 values. A large population. (From these few values, v seems to be positive but rather variable.)

(b) Obtain a suitable plot of your population. What do you notice?

Solution:

One quantitative variable, so a histogram. The population is large, so you can use more bins than usual. Sturges' rule says 14 bins (the logarithm below is base 2, or, the next power of 2 above 10,000 is 16,384 which is 2^{14}):

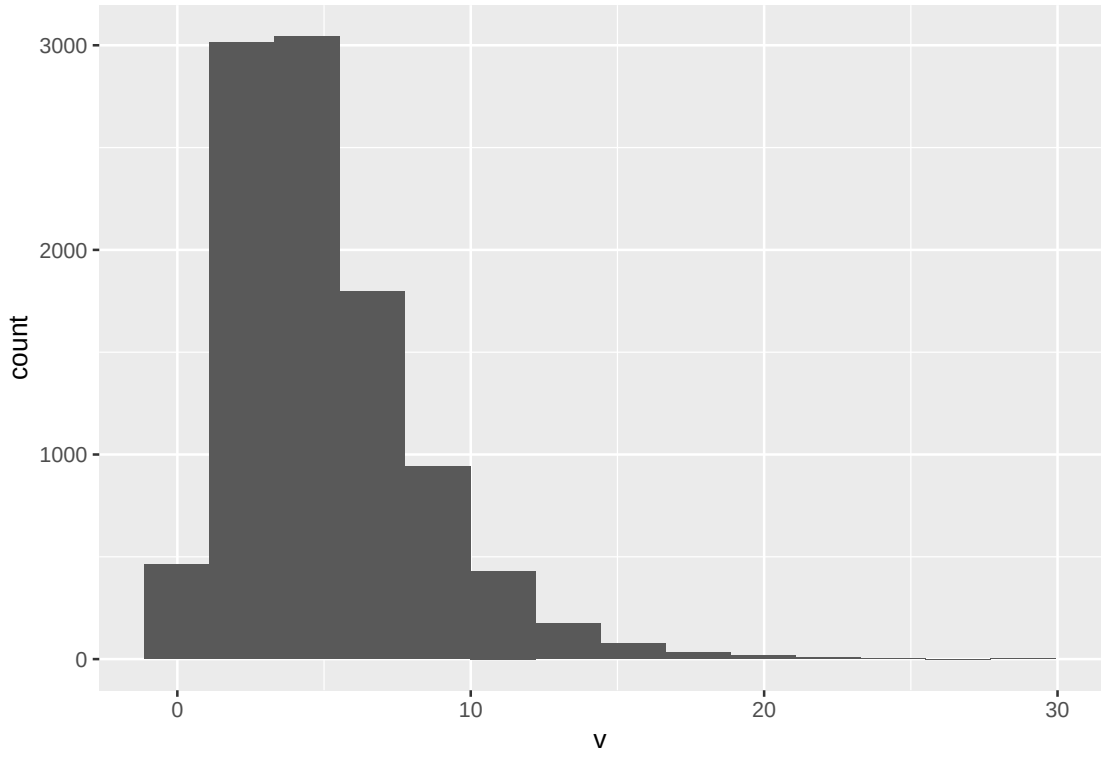
```
log(10000, 2)
```

```
## [1] 13.28771
```

```
2^14
```

```
## [1] 16384
```

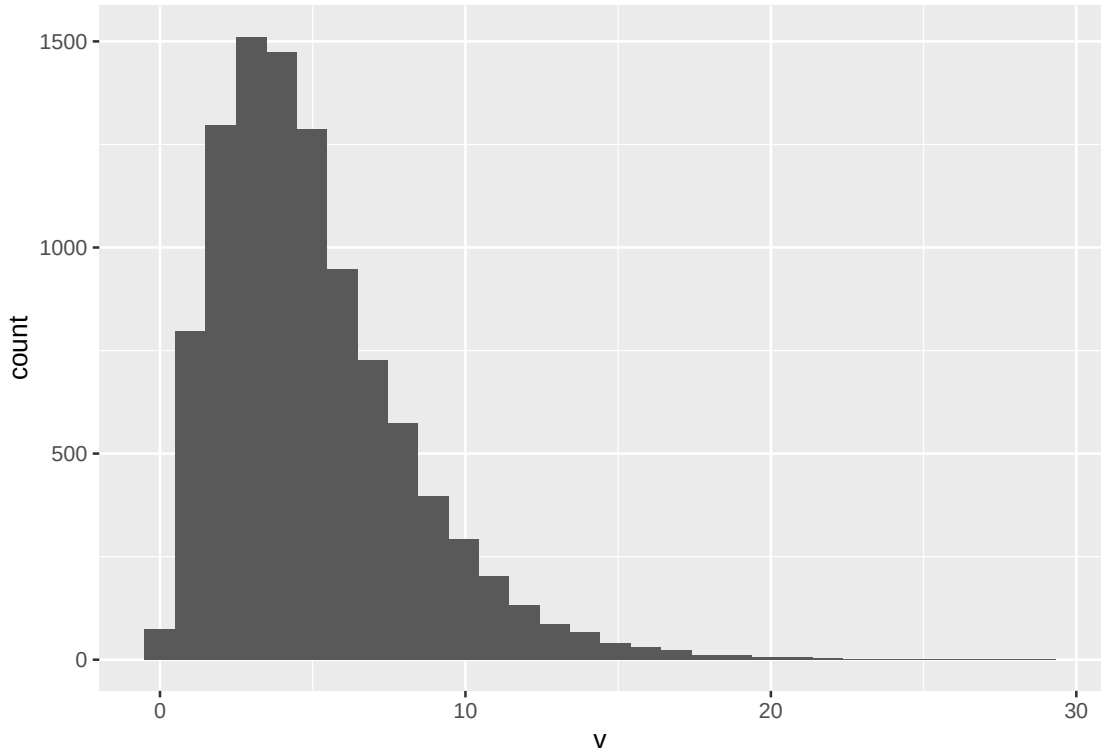
```
ggplot(pop, aes(x=v)) + geom_histogram(bins=14)
```



Pick a number of bins: the default 30 bins is pretty much always too many. Any number of bins that shows this shape is good as an answer, but you also need to display some thinking about how many bins to use, either starting with a rule as I did, or experimenting with different numbers of bins. Rules are not hard and fast; it so happened that I liked the picture that 14 bins gave, so I stopped there. Thirty bins, the default, is actually not bad here:

```
ggplot(pop, aes(x=v)) + geom_histogram()
```

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```

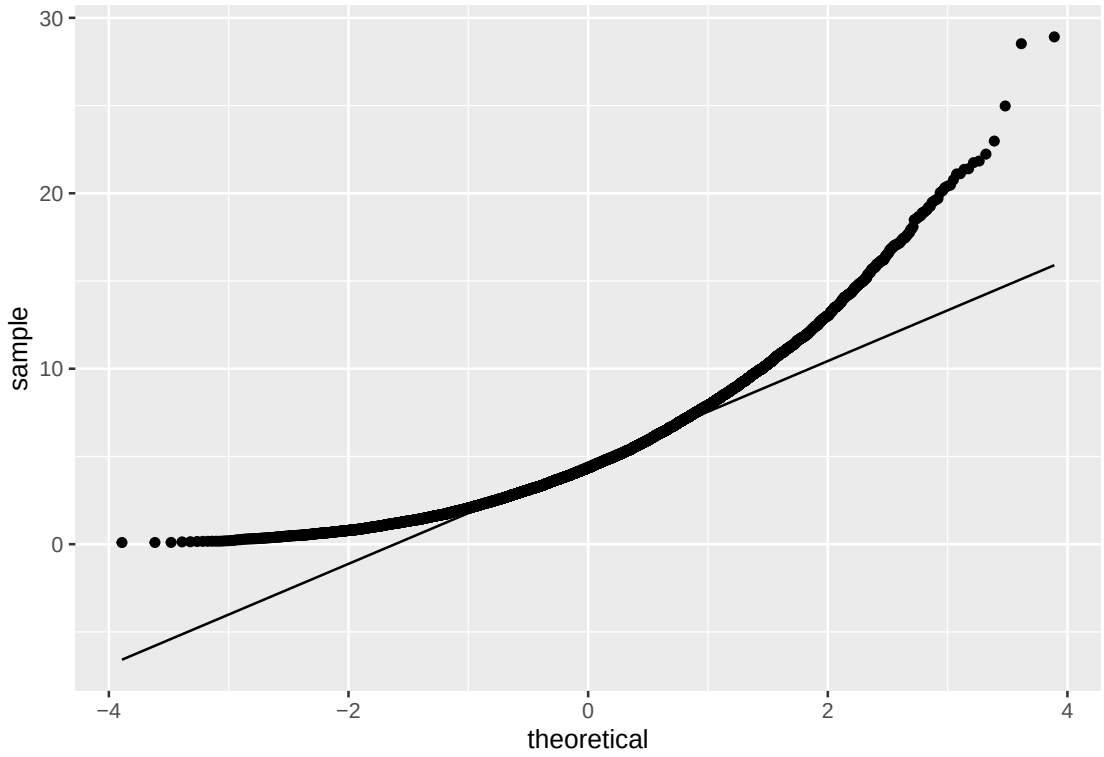


but if you do this, you need to say something that indicates some conscious thought, such as saying "this number of bins gives a good picture of the shape of the distribution", which I am OK with. *Have a reason for doing what you do.*

This is skewed to the right, or has a long right tail. This is a better description than "outliers": there are indeed some very large values (almost invisible on the histogram), but to say that is to imply that the rest of the distribution apart from the outliers has a regular shape, not something you can say here.[1]

Extra: The issue that's coming up is whether this is normally-distributed, which of course it is not. This is a normal quantile plot, which comes up in lecture right around the time you are reading this. (Idea: if the points follow the line, at least approximately, the variable is normally distributed; if not, not.):

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



That is your archetypal skewed-to-right. The two highest values are not a lot higher than the rest, again supporting a curved shape overall (skewness) on this plot, rather than the problem being outliers. (The highest values are consistent with the shape of the curve, rather than being unusually high compared to the curve.)

(c) If you take a sample of 10 observations from this population and run a t-test, how likely are you to (correctly) reject the null hypothesis $H_0 : \mu = 4$, against the alternative $H_a : \mu > 4$? Investigate by simulation.

Solution:

As you noted, this is a one-sided alternative, so make sure your code does the right thing. Take a lot of random samples, run the t-test on each one, grab the P-value each time, make a dataframe, count the number of P-values less or equal to your α:

```
rerun(1000, sample(pop$v, 10)) %>%
  map(~t.test(., mu=4, alternative = "greater")) %>%
  map_dbl("p.value") -> pvals
tibble(pvals) %>% count(pvals<=0.05)
```

```
## # A tibble: 2 x 2
##   `pvals <= 0.05`     n
## * <lgl>           <int>
## 1 FALSE             810
## 2 TRUE              190
```

The estimated power is only about 0.19.

Here, and elsewhere in this question, use at least 1000 simulations. More will give you more accurate results, but you'll have to wait longer for it to run. Your choice.

As a final remark, you *can not* do this one by algebra, as you might have done in other courses, because you do not know the functional form of the population distribution. The power calculations you may have done before as calculations typically assume a normal population, because if you don't, the algebra gets too messy too fast. (You'd need to know the distribution of the test statistic under the *alternative* hypothesis, which in cases beyond the normal is not usually known.)

(d) Try again with a sample size of 50 (leaving everything else the same). Explain briefly why the results so far are as you'd expect.

Solution:

For the code, this is copy-paste-edit. Just change the sample size:

```
rerun(1000, sample(pop$v, 50)) %>%
  map(~t.test(., mu=4, alternative = "greater")) %>%
  map_dbl("p.value") -> pvals
tibble(pvals) %>% count(pvals<=0.05)
```

```
## # A tibble: 2 x 2
##   `pvals <= 0.05`     n
## * <lgl>           <int>
## 1 FALSE             268
## 2 TRUE              732
```

The power is now much bigger, around 73%. This is as expected because with a larger sample size we should be more likely to reject a false null hypothesis.

The reason for this is that the mean of a bigger sample should be closer to the population mean, because of the Law of Large Numbers, and thus further away from the incorrect null hypothesis and more likely far enough away to reject it. In this case, as you will see shortly, the population mean is 5, and so, with a bigger sample, the sample mean will almost certainly be closer to 5 and further away from 4.

I have a feeling you could formalize this kind of argument with Chebyshev's inequality, which would apply to any kind of population.[2] I think I'd have to write it down to get it right, though.

(e) Again by simulation, estimate the probability that the null hypothesis $H_0 : \mu = 5$ will be rejected when a sample of size 10 is taken from this population, in favour of the alternative $H_a : \mu > 5$. Explain briefly why the answer is not what you would have expected, and why that happened here. (Hint: what is the population mean?)

Solution:

Taking the hint first:

```
pop %>%
  summarize(m = mean(v))
```

```
## # A tibble: 1 x 1
##       m
```

```
##   <dbl>
## 1  5.00
```

(I'm hoping that some light dawns at this point), and copy-paste-edit your simulation code again:

```
rerun(1000, sample(pop$v, 10)) %>%
  map(~t.test(., mu=5, alternative = "greater")) %>%
  map_dbl("p.value") -> pvals
tibble(pvals) %>% count(pvals<=0.05)
```

```
## # A tibble: 2 x 2
##   `pvals <= 0.05`     n
## * <lgl>           <int>
## 1 FALSE             980
## 2 TRUE               20
```

The "power" is estimated to be 0.020. (Again, your value won't be exactly this, most likely, but it should be somewhere close.)

So what were we expecting? This time, the null hypothesis, that the population mean is 5, is actually *true*. So rejecting it is now a *type I error*, and the probability of that should be α, which was 0.05 here. In our simulation, though, the estimated probability is quite a bit *less* than 0.05. (Your result will probably differ from mine, but it is not likely to be bigger than 0.05).

To think about why that happened, remember that this is a very skewed population, and the sample size of 10 is not big, so this is not really the situation in which we should be using a t-test. The consequence of doing so anyway, which is what we investigated, is that the actual α of our test is not 0.05, but something smaller: the test is not properly calibrated.

If you do this again for a sample of size 50, you'll find that the simulation tells you that α is closer to 0.05, but still less. The population is skewed enough that the Central Limit Theorem still hasn't kicked in yet, and so we still cannot trust the t-test to give us a sensible P-value.

Extra: a lot more discussion on what is happening here:

This test is what is known in the jargon as "conservative". To a statistician, this means that the probability of making a type I error is smaller than it should be. That is in some sense safe, in that if you reject, you can be pretty sure that this rejection is correct, but it makes it a lot harder than it should to reject in the first place, and thus you can fail to declare a discovery when you have really made one (but the test didn't say so).

I did some investigation to see what was going on. First, I ran the simulation again, but this time keeping the mean and SD of each sample, as well as the t-statistic:

```
rerun(1000, sample(pop$v, 10)) %>%
  enframe(name="sim") %>%
  unnest(value) %>%
  group_by(sim) %>%
  summarize(xbar=mean(value), s=sd(value), tstat=(xbar-5)/(s/sqrt(10))) -> mean_sd
mean_sd
```

```
## # A tibble: 1,000 x 4
##      sim  xbar     s   tstat
##  * <int> <dbl> <dbl>   <dbl>
##  1     1  3.67  1.10 -3.84
```

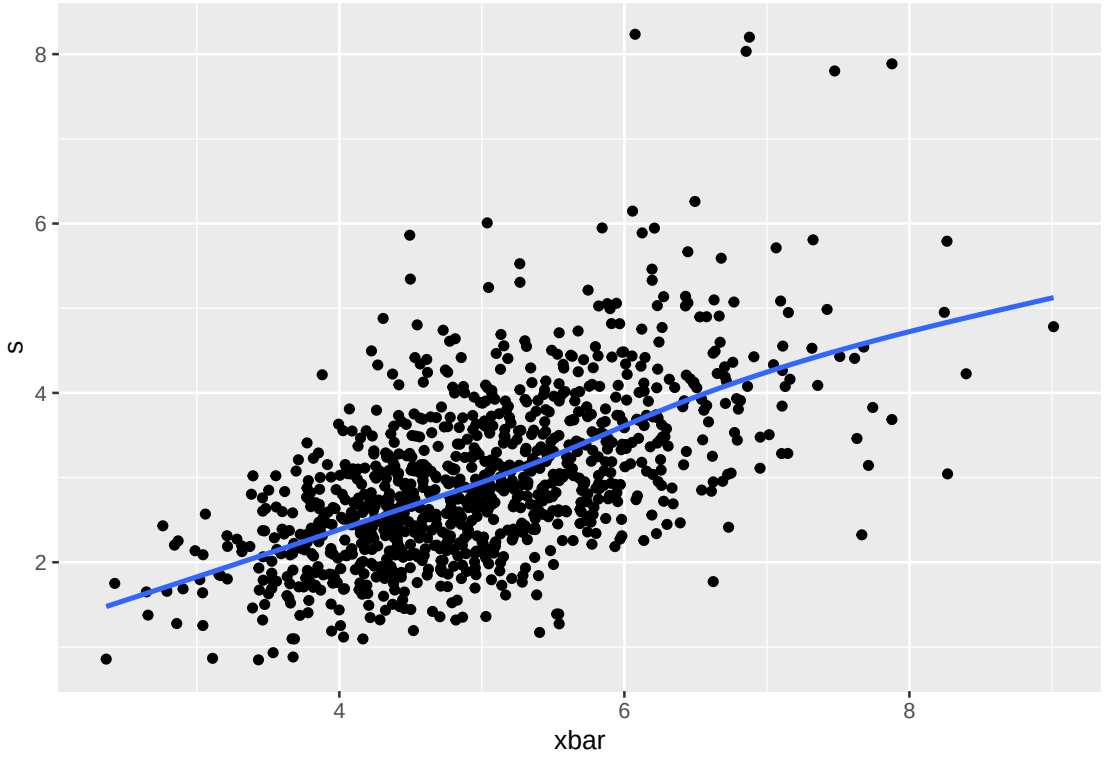
```
##  2     2  6.62  1.77  2.90
##  3     3  6.77  5.07  1.10
##  4     4  9.01  4.78  2.65
##  5     5  3.38  2.80 -1.82
##  6     6  6.08  2.74  1.25
##  7     7  4.81  4.00 -0.151
##  8     8  5.09  3.93  0.0736
##  9     9  4.59  2.41 -0.545
## 10    10  5.43  3.51  0.385
## # ... with 990 more rows
```

As for coding, I turned the simulated samples into a dataframe, got out the individual values, obtaining a column `sim` that numbers the individual samples, and then was able to get what I wanted by `group_by` and `summarize` as usual. Try it one line at a time to see how it works. The `enframe` turns a `list` (which is what comes out of `rerun`) into a dataframe. I'm never quite sure how it will come out, but I try it first and then deal with what it produces. When I was working this out, I started with a `rerun(2, ...)` so that I could see what it was doing, and when I was happy that it was all working, I did it again with the final number of simulations.

After that, I played around with several things, but I found something interesting when I plotted the sample mean and SD *against each other*:

```
ggplot(mean_sd, aes(x=xbar, y=s)) + geom_point() + geom_smooth(se=F)
```

```
## `geom_smooth()` using method = 'gam' and formula 'y ~ s(x, bs = "cs")'
```



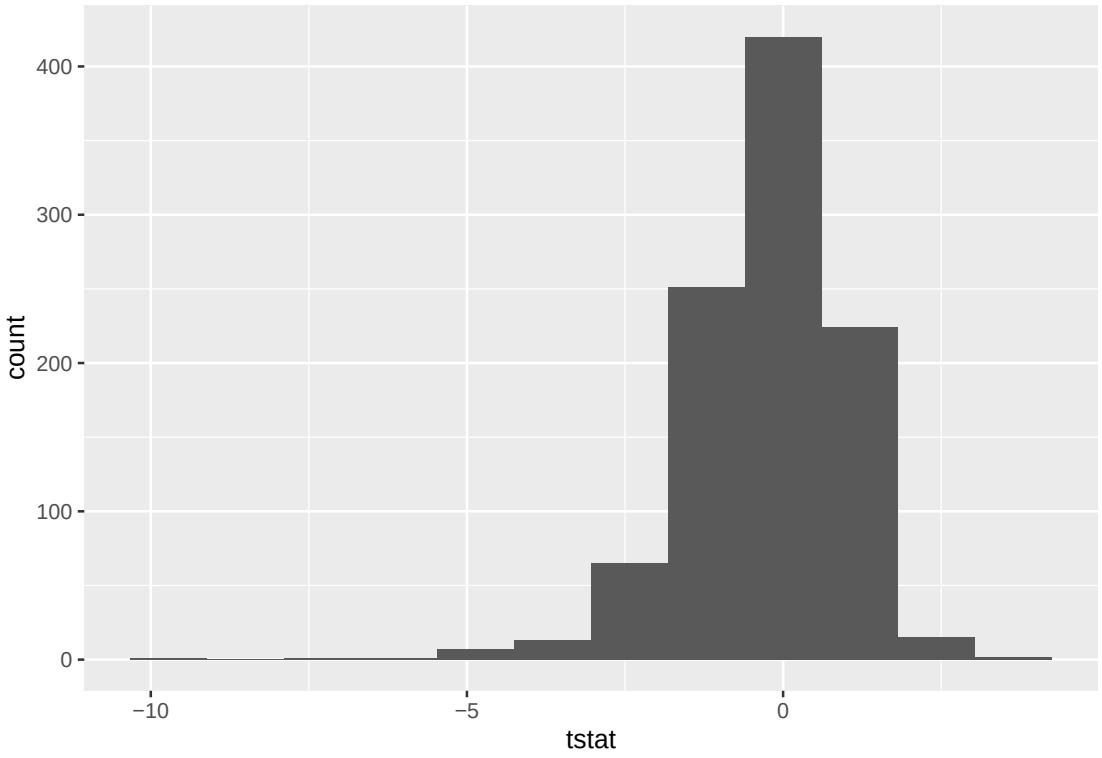
When the sample mean is bigger, so is the sample standard deviation!

This actually does make sense, if you stop to think about it. A sample with a large mean will have some of those values from the long right tail in it, and having those values will also make

the sample more spread out. The same does not happen at the low end: if the mean is small, all the sample values must be close together and the SD will be small also.[3]

It wasn't clear to me what that would do to the t-statistic. A larger sample mean would make the top of the test statistic bigger, but a larger sample mean would also go with a larger sample SD, and so the bottom of the test statistic would be bigger as well. That's why I included this in the simulation too:

```
ggplot(mean_sd, aes(x=tstat)) + geom_histogram(bins=12)
```



Well, well. Skewed to the *left*.

This too makes sense with a bit of thought. A small sample mean will also have a small sample SD, so the test statistic could be more negative. But a large sample mean will have a *large* sample SD, so the test statistic won't get so positive. Hence, in our simulation, the test statistic won't get large enough to reject with as often as it should. Thus, the type I error probability that is too small.

(there is a second question on the next page)

2. A company produces prepackaged diet meals. These meals are advertised as containing "6 ounces of protein per package". A consumer organization is concerned that this is not accurate. The organization takes a random sample of 20 of these meals, and measures the protein content of each one. The data are in http://ritsokiguess.site/STAC33/protein.txt as one column.

   (a) Read in and display (some of) the data.

Solution:

The usual. This is one column only, so you can pretend the columns are separated by anything at all and it will still work:

```
my_url <- "http://ritsokiguess.site/STAC33/protein.txt"
meals <- read_table(my_url)
```

```
##
## -- Column specification --------------------------------------------------------
## cols(
##   protein = col_double()
## )
```

```
meals %>% arrange(protein)
```

```
## # A tibble: 20 x 1
##    protein
##      <dbl>
##  1     3.6
##  2     4.2
##  3     4.3
##  4     4.4
##  5     4.7
##  6     4.9
##  7     5.1
##  8     5.1
##  9     5.2
## 10     5.5
## 11     5.5
## 12     5.6
## 13     5.7
## 14     5.8
## 15     5.8
## 16     6.1
## 17     6.1
## 18     6.1
## 19     6.1
## 20     6.1
```

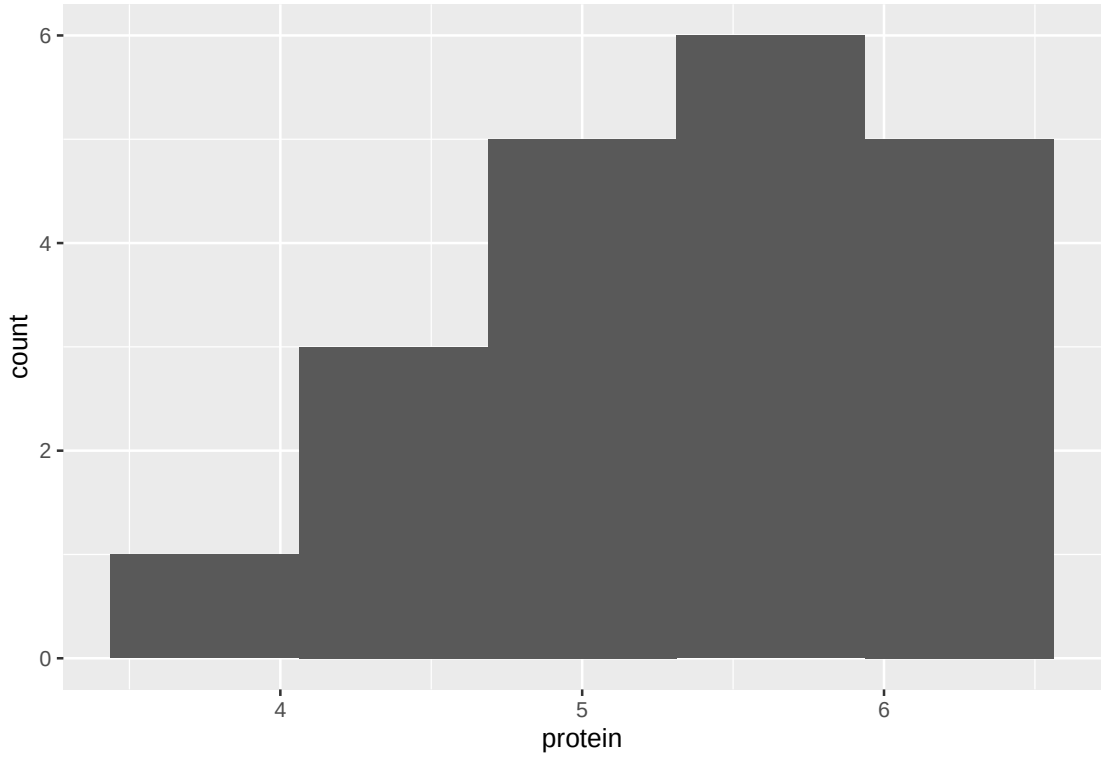
Get it to work via one of the methods you've seen in this class (ie., *not* `read.table`); I don't mind how you manage it.

   (b) Make a suitable graph of your data.

Solution:

One quantitative variable, so a histogram with a sufficiently small number of bins:

```
ggplot(meals, aes(x=protein)) + geom_histogram(bins = 5)
```



(c) Why might a sign test be better than a t-test for assessing the average amount of protein per package? Explain briefly. ("Average" here means any measure of centre.)

Solution:

The shape of the above distribution is skewed to the left, and not symmetric like a normal distribution. (If you say "the histogram is not normal", make sure you also say how you know.) This means that the median would be a better measure of "average" (that is, centre) than the mean is, because the mean would be pulled downwards by the long tail, and the median would not. To complete the story, the sign test is a test of the median, so the sign test would be better than the t-test, which is a test of the mean.

The other thing you might consider is the sample size, 20, which *might* be large enough to overcome this amount of skewness, but then again it might not be. So you could say that we should be cautious and run the sign test here instead.

(d) Run a suitable sign test for these data. What do you conclude?

Solution:

First, if you have not already done so, install `smmr` *following the instructions in the lecture notes*. (This one is not just `install.packages`.) Then, make sure you have a `library(smmr)` somewhere above where you are going to use something from it. Once that is in place, remember

what we were interested in: was the median protein content 6 ounces, or is there evidence that it is something different? (The "not accurate" in the question says that the median could be higher or lower, either of which would be a problem, and so we need a two-sided alternative.) Thus, the null median is 6 and we need a two-sided test, which goes this way:

```
sign_test(meals, protein, 6)
```

```
## $above_below
## below above
##    15     5
##
## $p_values
##   alternative    p_value
## 1       lower 0.02069473
## 2       upper 0.99409103
## 3   two-sided 0.04138947
```

The P-value, 0.0414, is less than 0.05, so we reject the null hypothesis and conclude that the median is different from 6 ounces. The advertisement by the company is not accurate.

Make sure you give the actual P-value you are comparing with 0.05, since otherwise your answer is incomplete. That is, you need to say more than just "the P-value is less than 0.05"; there are three P-values here, and only one of them is the right one.

Extra: we already decided that a t-test is not the best here, but I am curious as to how different its P-value is:

```
with(meals, t.test(protein, mu=6))
```

```
##
##  One Sample t-test
##
## data:  protein
## t = -4.2312, df = 19, p-value = 0.000452
## alternative hypothesis: true mean is not equal to 6
## 95 percent confidence interval:
##  4.946263 5.643737
## sample estimates:
## mean of x
##     5.295
```

The conclusion is the same, but the P-value is a lot smaller. I don't think it should really be this small; this is probably because the mean is pulled down by the left skew and so really ought not to look so far below 6. I am inclined to think that if the t-test were correct, its P-value ought to be between this and the one from the sign test, because the t-test uses the actual data values, and the sign test uses the data less efficiently (only considering whether each one is above or below the null median).

(e) In your sign test, how could you have deduced that the P-value was going to be small even without looking at any of the P-values themselves? Explain briefly.

Solution:

Look at the other part of the output, the count of values above and below the null median.

(You might have to click on "R Console" to see it.) If the null hypothesis was correct and the median was really 6, you'd expect to see about half the data values above 6 and about half below. But that is not what happened: there were 15 values below and only 5 above. Such an uneven split is rather unlikely if the null hypothesis was correct. So we would guess that our P-value would be small, as indeed it is.

(f) Obtain a 90% confidence interval for the population median protein content. What does this tell you about the reason for the rejection or non-rejection of the null hypothesis above?

Solution:

This is `ci_median`, but you need to be paying attention: it's not the default 95% confidence level, so you have to specify that as well:

```
ci_median(meals, protein, conf.level = 0.90)
```

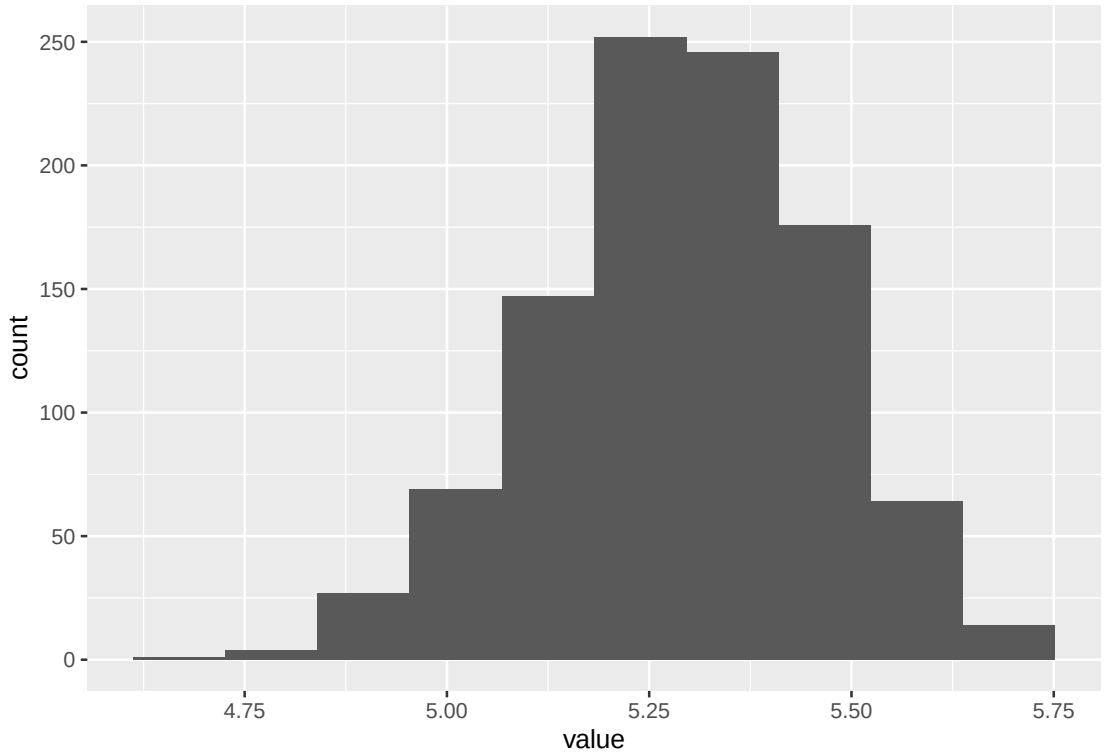
```
## [1] 4.905273 5.793750
```

The interval goes from 4.91 to 5.79. (The data values have one decimal place, so you could justify two decimals in the CI for the median, but anything beyond that is noise and you shouldn't give it in your answer.[4])

This interval is entirely below 6 (the null median), so evidently the reason that we rejected 6 as the population median is that the actual population median is *less than* 6.

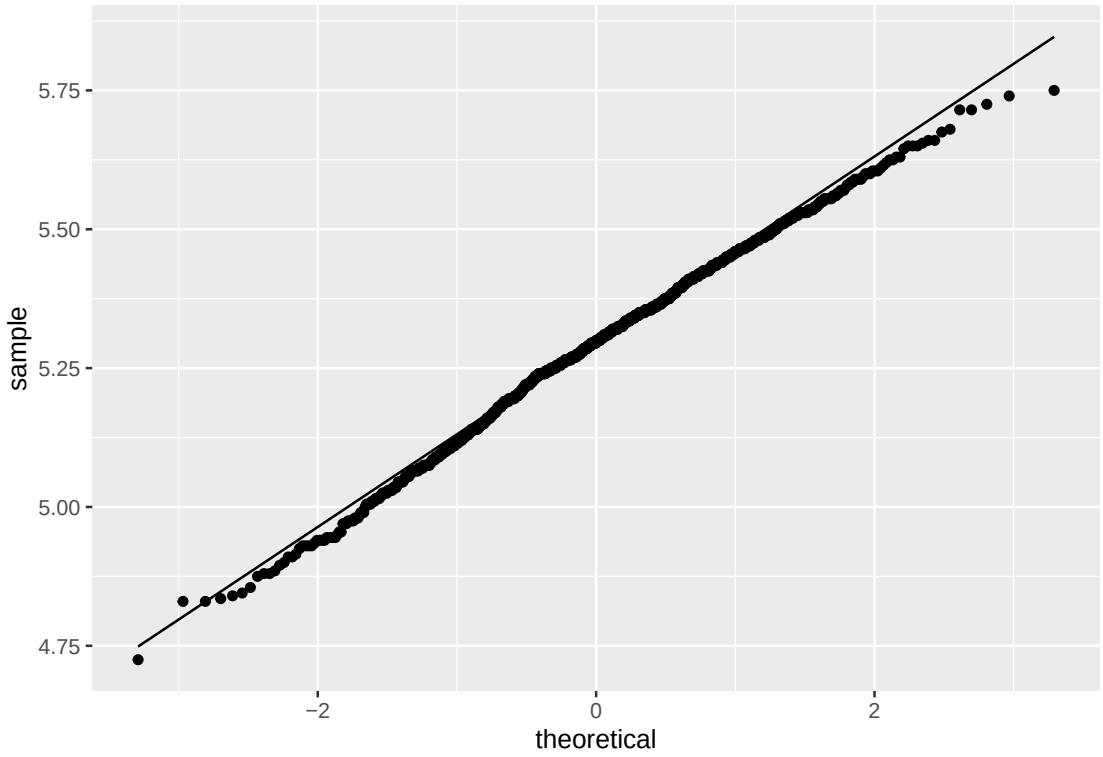
Extra: the CI for the median is not that different from the one for the mean, which suggests that maybe the t-test was not so bad after all. If you want to investigate further, you can try finding a bootstrapped sampling distribution of the sample mean, and see how non-normal it looks:

```
rerun(1000, sample(meals$protein, replace = TRUE)) %>%
  map_dbl(~mean(.)) %>%
  enframe() -> d
ggplot(d, aes(x = value)) + geom_histogram(bins = 10)
```



That is pretty close to normal. So the t-test would in actual fact have been fine. To confirm, a normal quantile plot:

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



A *tiny bit* skewed to the left.

But I didn't ask you to do this, because I wanted to give you a chance to do a sign test for what seemed like a good reason.

Extra 2: I mentioned in an endnote that the endpoints of the CI for the median are actually data points, only we didn't see it because of the accuracy to which `ci_median` was working. You can control this accuracy by an extra input `tol`. Let's do something silly here:

```
ci_median(meals, protein, conf.level = 0.90, tol = 0.00000001)
```

```
## [1] 4.900001 5.799999
```

This takes a bit longer to run, since it has to get the answer more accurately, but now you can see how the interval goes from "just over 4.9" to "just under 5.8", and it actually makes the most sense to give the interval as "4.9 to 5.8" without giving any more decimals.

Extra 3: the reason for the confidence interval endpoints to be data values is that the interval comes from inverting the test: that is, finding the values of the population median that would not be rejected by a sign test run on our data. Recall how the sign test works: it is based on a count of how many data values are above and below the hypothesized population median. These counts are only going to change as the hypothesized median changes if you hit a data point, since that's the only way you can change how many values are above and below.[5] Thus, the only places where changing the null median changes whether or not a value for it is inside or outside the confidence interval are at data values, and thus the ends of the interval must be at (or, perhaps more strictly, just above or below) data values.

This is a peculiarity of using the sign test to get a CI for the median. If, say, you were to invert the t-test to get a confidence interval for the mean, you wouldn't see that. (This is in fact exactly what you do to get a confidence interval for the mean, but this is not the way it

is usually introduced.) The reason that the CI for the mean (based on the t-test) is different from the one for the median (based on the sign test) is that if you change the null hypothesis in the t-test, however slightly, you change the P-value (maybe only slightly, but you change it). So the CI for the mean, based on the t-test, is not required to have data points at its ends, and indeed usually does not. The difference is in the kind of distribution the test statistic has; the t-distribution is continuous, while the sign test statistic (a count of the number of values above or below something) is discrete. It's the discreteness that causes the problems.

Notes

1. The question to ask yourself is whether the shape comes from the entire distribution, as it does here (skewness), or whether it comes from a few unusual observations (outliers).

2. It has to have a standard deviation, though, which rules out Cauchy, but our population seems well-enough behaved to have a standard deviation.

3. You might have something lurking in your mind about the sample mean and sample SD/variance being *independent*, which they clearly are not here. That is true if the samples come from a *normal distribution*, and from that comes independence of the top and bottom of the t-statistic. But here is an example of how everything fails once you go away from normality, and how you have to rely on the central limit theorem, or large sample sizes more generally, for most of your theory to be any good.

4. There is actually slightly more to it here: the ends of this confidence interval for the median are always data values, because of the way it is constructed, so the actual end points really ought to be given to the *same* number of decimals as the data, here 4.9 to 5.8. The output given is not exactly 4.9 and 5.8 because of inaccuracy in the bisection.

5. There is a technicality about what happens when the null median is exactly equal to a data value; see PASIAS for more discussion on this.