Assignment 3

Instructions (the same as for Assignment 1): Make an R Notebook and in it answer the two questions below (one Notebook for both questions). 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. The only reason to contact the instructor while working on the assignments is to report something missing like a data file that cannot be read. Seeking or obtaining help from anyone else is **an academic offence**.

You have 3 hours to complete this assignment after you start it.

My solutions to this assignment, with extra discussion, will be available after everyone has handed in their assignment.

Finally, you will need to install and load the `smmr` package, following the instructions in lecture, along with loading the `tidyverse` as usual. If you have already installed `smmr`, you will have more time to work on this assignment.

1. You are designing a study to test the null hypothesis that a population mean is 0 against the alternative hypothesis that it is greater than 0. Assume that the population SD is $\sigma = 15$. It is important to detect the alternative $\mu = 2$; that is, we want to design the study so that most of the time the null hypothesis would be (correctly) rejected if in fact $\mu = 2$. A one-sample t-test will be used, and the data values are assumed to have a normal distribution.

   (a) Use simulation to estimate the power of this test when the sample size is 100. Use $\alpha = 0.05$.

   Solution:

   This is `rerun`. Use at least 1000 simulations (more, if you're willing to wait for it). In `rnorm`, the sample size is first, then the (true) population mean, then the (assumed) population SD:

```
rerun(1000, rnorm(100, 2, 15)) %>%
  map(~t.test(., mu=0, 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             639
## 2 TRUE              361
```

The power is (estimated as) a disappointing 0.361. Your answer won't (most likely) be the same as this, but it should be somewhere close. I would like to see you demonstrate that you know what power is, for example "if the population mean is actually 2, the null hypothesis $H_0 : \mu = 0$, which is wrong, will only be rejected about 36% of the time".[1]

The test we are doing is one-sided, so you need the `alternative` in there. If you omit it, you'll have the answer to a different problem:

```
rerun(1000, rnorm(100, 2, 15)) %>%
  map(~t.test(., mu=0)) %>%
  map_dbl("p.value") -> pvals
tibble(pvals) %>% count(pvals<=0.05)
```

```
## # A tibble: 2 x 2
##   `pvals <= 0.05`     n
##   <lgl>           <int>
## 1 FALSE             740
## 2 TRUE              260
```

This is the probability that you reject $H_0 : \mu = 0$ in favour of $H_a : \mu \neq 0$. This is smaller, because the test is "wasting effort" allowing the possibility of rejecting when the sample mean is far enough *less* than zero, when most of the time the samples drawn from the true distribution have mean greater than zero. (If you get a sample mean of 2.5, say, the P-value for a one-sided test will be smaller than for a two-sided one.)

Extra 1:

This low power of 0.361 is because the population SD is large relative to the kind of difference from the null that we are hoping to find. To get a sense of how big the power might be, imagine you draw a "typical" sample from the true population: it will have a sample mean of 2 and a sample SD of 15, so that t will be about

```
(2-0)/(15/sqrt(100))
```

```
## [1] 1.333333
```

You won't reject with this (t would have to be bigger than 2), so in the cases where you *do* reject, you'll have to be more lucky: you'll need a sample mean bigger than 2, or a sample SD smaller than 15. So the power won't be very big, less than 0.5, because about half the time you'll get a test statistic less than 1.33 and about half the time more, and not all of *those* will lead to rejection.

Extra 2:

This is exactly the situation where `power.t.test` works, so we can get the exact answer (you need all the pieces):

```
power.t.test(n=100, delta=2-0, sd=15, type="one.sample",
             alternative = "one.sided")
```

```
##
##      One-sample t test power calculation
##
##               n = 100
##           delta = 2
##              sd = 15
##       sig.level = 0.05
```

```
##           power = 0.3742438
##     alternative = one.sided
```

Your answer, from 1000 simulations, should be within about 3 percentage points of that. (Mine was only about 1 percentage point off.)

(b) Again by using simulation, estimate how large a sample size would be needed to obtain a power of 0.80. Show and briefly explain your process.

Solution:

The point of this one is the process as well as the final answer, so you need to show and justify what you are doing. Showing only the final answer *does not* show that you know how to do it. The *whole point* of this one is to make mistakes and fix them!

The simulation approach does not immediately give you a sample size for fixed power, so what you have to do is to try different sample sizes until you get one that gives a power close enough to 0.80. You have to decide what "close enough" means for you, given that the simulations have randomness in them. I'm going to use 10,000 simulations for each of my attempts, in the hope of getting a more accurate answer.

First off, for a sample size of 100, the power was too small, so the answer had better be bigger than 100. I'll try 200. For these, copy and paste the code, changing the sample size each time:

```
rerun(10000, rnorm(200, 2, 15)) %>%
  map(~t.test(., mu=0, 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            4064
## 2 TRUE             5936
```

A sample size of 200 isn't big enough yet. I'll double again to 400:

```
rerun(10000, rnorm(400, 2, 15)) %>%
  map(~t.test(., mu=0, 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            1582
## 2 TRUE             8418
```

Getting closer. 400 is too big, but closer than 200. 350?

```
rerun(10000, rnorm(350, 2, 15)) %>%
  map(~t.test(., mu=0, 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            1932
## 2 TRUE             8068
```

Close! I reckon you could call that good (see below), or try again with a sample size a bit less than 350:

```
rerun(10000, rnorm(340, 2, 15)) %>%
  map(~t.test(., mu=0, 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            2153
## 2 TRUE             7847
```

340 is definitely too small:

```
rerun(10000, rnorm(347, 2, 15)) %>%
  map(~t.test(., mu=0, 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            2045
## 2 TRUE             7955
```

This is actually not as close as I was expecting. I think we are getting close to simulation accuracy for this number of simulations. If we do 10,000 simulations of an event with probability 0.8 (correctly rejecting this null), below are the kind of results we might get.[2] This is the middle 95% of that distribution.

```
qbinom(c(0.025,0.975), 10000, 0.8)
```

```
## [1] 7921 8078
```

Anything between those limits is the kind of thing we might get by chance, so simulation doesn't let us distinguish between 347 and 350 as the correct sample size. Unless we do more than 10,000 simulations, of course!

If you stuck with 1000 simulations each time, these are the corresponding limits:

```
qbinom(c(0.025,0.975), 1000, 0.8)
```

```
## [1] 775 824
```

and any sample sizes that produce an estimated power between these are as accurate as you'll get. (Here you see the advantage of doing more simulations.)

If you've been using 10,000 simulations each time like me, you'll have noticed that these actually take a noticeable time to run. This is why coders always have a coffee or something else to sip on while their code runs; coders, like us, need to see the output to decide what to do next. Or

you could install the beepr package, and get some kind of sound when your simulation finishes, so that you'll know to get off Twitter[3] and see what happened. There are also packages that will send you a text message or will send a notification to all your devices.

What I want to see from you here is some kind of trial and error that proceeds logically, sensibly increasing or decreasing the sample size at each trial, until you have gotten reasonably close to power 0.8.

Extra: once again we can figure out the correct answer:

```
power.t.test(power = 0.80, delta=2-0, sd=15, type="one.sample",
             alternative = "one.sided")
```

```
##
##      One-sample t test power calculation
##
##               n = 349.1256
##           delta = 2
##              sd = 15
##       sig.level = 0.05
##           power = 0.8
##     alternative = one.sided
```

This does not answer the question, though, since you need to do it by simulation with trial and error. If you want to do it this way, do it at the *end* as a check on your work; if the answer you get this way is very different from the simulation results, that's an invitation to check what you did.

350 actually *is* the correct answer. But you will need to try different sample sizes until you get close enough to a power of 0.8; simply doing it for $n = 350$ is not enough, because how did you know to try 350 and not some other sample size?

2. Ben Roethlisberger plays (American) football for the Pittsburgh Steelers. He plays as a quarterback, which means that his job is to throw (pass) the ball so that one of his teammates can catch it. Each time he makes a pass that is caught, this is called a "completion", and the team coaches are interested in his average number of completions per game (this average could be the mean or the median).

   In 2010, Roethlisberger was suspended for the first four games of the season, and there was concern that this might affect his performance (in terms of the number of passes completed in the games after he returned). The Pittsburgh Steelers did not play in week 5 of the 2010 season; the season is 17 weeks long (one game per week) and each team has one week in which they do not play.

   The data are in http://ritsokiguess.site/STAC32/roethlisberger.csv. There are four columns: the year (always 2010), the week number of the season that the game was played in, the name of the opposing team, and the number of completed passes by Roethlisberger in the game.

   (a) Read in and display (some of) the data. Do you have what you were expecting?

Solution:

Reading in is the usual, noting that this is a `.csv`:

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

```
## Parsed with column specification:
```

```
## cols(
##   season = col_double(),
##   week = col_double(),
##   opponent = col_character(),
##   completed = col_double()
## )
```

```
ben
```

```
## # A tibble: 12 x 4
##    season  week opponent      completed
##     <dbl> <dbl> <chr>             <dbl>
##  1   2010     6 Cleveland            16
##  2   2010     7 Miami                19
##  3   2010     8 New Orleans          17
##  4   2010     9 Cincinnati           17
##  5   2010    10 New England          30
##  6   2010    11 Oakland              18
##  7   2010    12 Buffalo              20
##  8   2010    13 Baltimore            22
##  9   2010    14 Cincinnati           21
## 10   2010    15 New York Jets        23
## 11   2010    16 Carolina             22
## 12   2010    17 Cleveland            15
```

Since "Roethlisberger" is a lot to type every time, I called the dataframe by his first name.

I am showing all 12 rows here; you are probably seeing only 10, and will have to scroll down to see the last two.

I have the four variables promised, and I also have a sensible number of rows. In particular, there is no data for weeks 1–4 (the suspension) and for week 5 (in which the team did not play), but there is a number of passes completed for all the other weeks of the season up to week 17. (If Roethlisberger had not played in any other games, you can expect that I would have told you about it.)

Extra: I did some processing to get the data to this point. I wanted to ask you about the 2010 season, and that meant having the 2009 data to compare it with. So I went here, scrolled down to Schedule and Game Results, and clicked on each of the Boxscores to get the player stats by game. Then I made a note of the opponent and the number of passes completed, and did the same for 2010. I put them in a file I called `r1.txt`, in aligned columns, and read that in. (An alternative would have been to make a spreadsheet and save that as a `.csv`, but I already had R Studio open.) Thus:

```
r0 <- read_table("r1.txt")
```

```
## Parsed with column specification:
## cols(
##   season = col_double(),
##   week = col_double(),
##   opponent = col_character(),
##   completed = col_double()
## )
```

```
r0
```

```
## # A tibble: 27 x 4
##    season  week opponent    completed
##     <dbl> <dbl> <chr>           <dbl>
##  1   2009     1 Tennessee          33
##  2   2009     2 Chicago            23
##  3   2009     3 Cincinnati         22
##  4   2009     4 San Diego          26
##  5   2009     5 Detroit            23
##  6   2009     6 Cleveland          23
##  7   2009     7 Minnesota          14
##  8   2009     9 Denver             21
##  9   2009    10 Cincinnati         20
## 10   2009    11 Kansas City        32
## # ... with 17 more rows
```

I was curious about the season medians (for reasons you see later), thus:

```
r0 %>% group_by(season) %>% summarise(med = median(completed))
```

```
## `summarise()` ungrouping output (override with `.groups` argument)
```

```
## # A tibble: 2 x 2
##   season   med
##    <dbl> <dbl>
## 1   2009  22
## 2   2010  19.5
```

You will realize that my asserted average for "previous seasons" is close to the median for 2009. Here is where I have to admit that I cheated. It actually *is* the median for 2009, except that there are some games in 2010 where Roethlisberger had 22 completed passes and I didn't want to mess the sign test up (I talk more about this later). So I made it 22.5, which is a possible value for the median of an even number of whole-number values.

Anyway, the last thing to do is to grab only the rows for 2010 and save them for you. This uses `filter` to select only the rows for which something is true (which we have seen several times in Extras, but meet "properly" later):

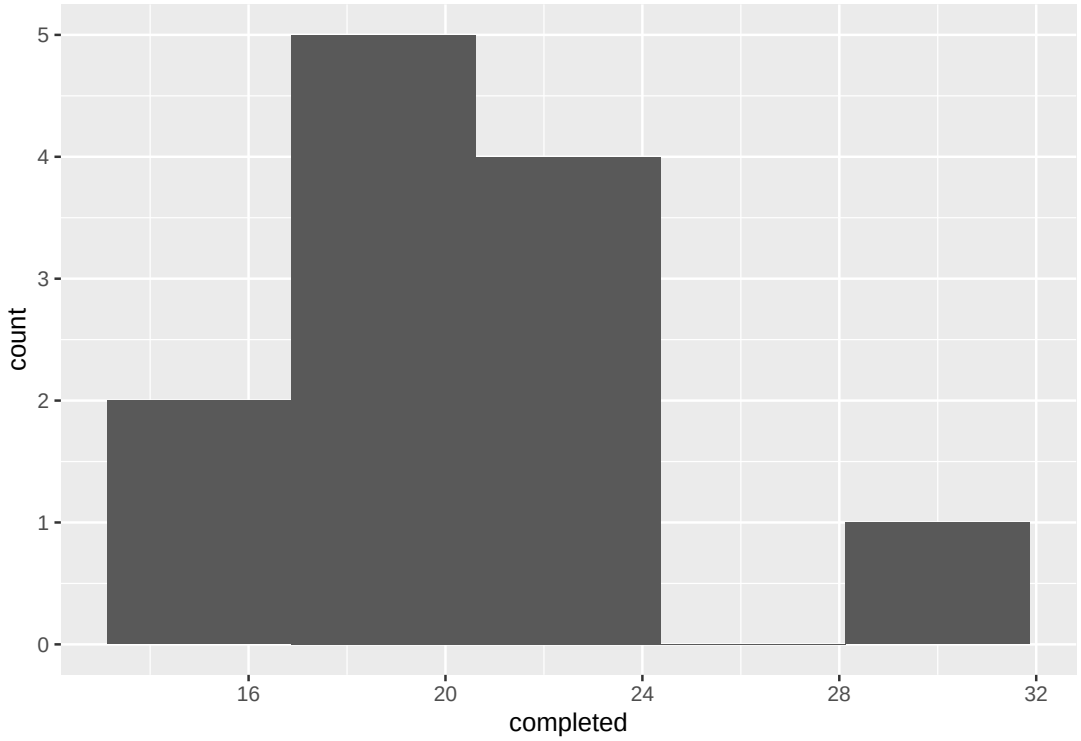
```
library(smmr)
r0 %>% filter(season==2010) -> r1
write_csv(r1, "roethlisberger.csv")
```

(b) Make a suitable graph of the number of completed passes, and explain briefly why you would have some doubts about using t-procedures in this situation.

Solution:

Don't be tempted to think *too* hard about the choice of graph (though I talk more about this below). One quantitative variable, so a histogram again. There are only 12 observations, so 5 bins is about as high as you should go:

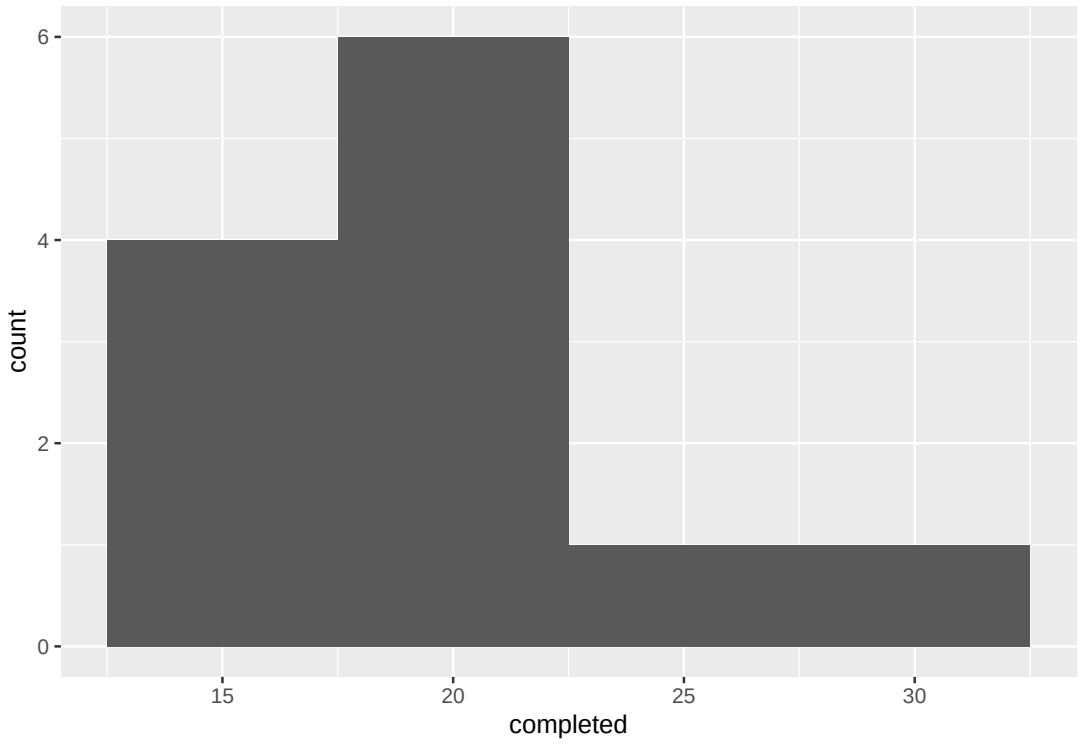
```
ggplot(ben, aes(x=completed)) + geom_histogram(bins=5)
```



This one shows an outlier: there is one number of completed passes that is noticeably higher than the rest. A normal distribution doesn't have outliers, and so this, coupled with a small sample in which normality is important, means that we should not be using a t-test or confidence interval.

If you chose a different number of bins, you might get a different look. Here's 4 bins:

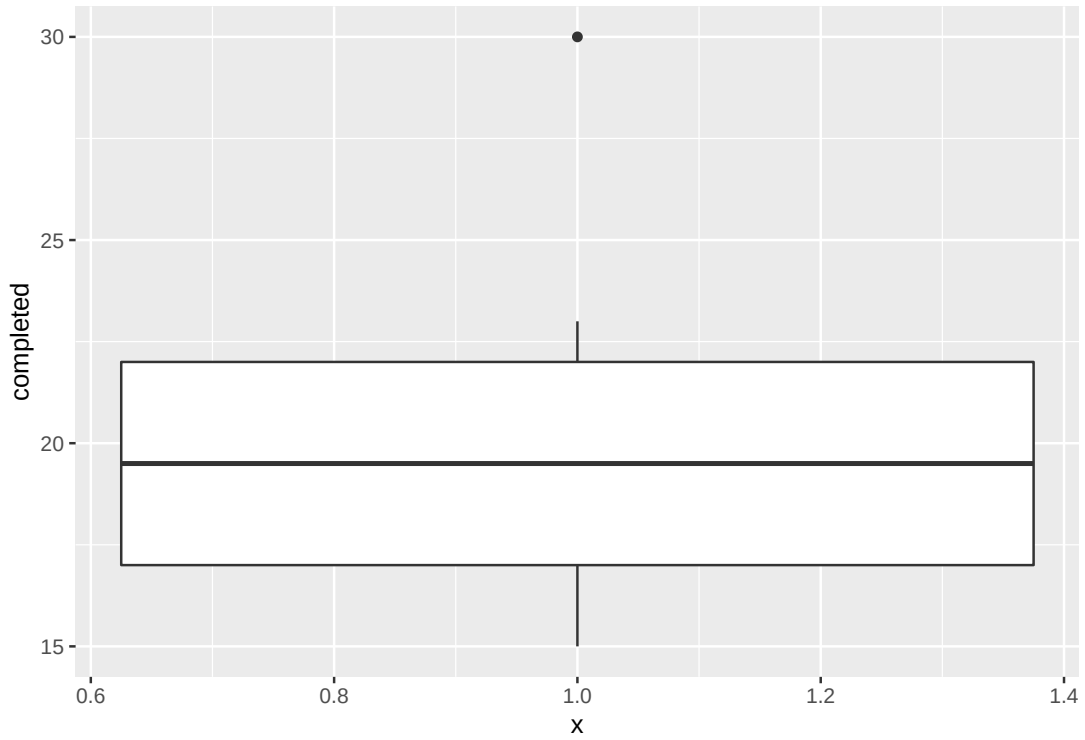
```
ggplot(ben, aes(x=completed)) + geom_histogram(bins=4)
```



That looks more like right-skewness, but the conclusion is the same.

Extra: if you have read, for example, Problem 6.1 in PASIAS, you'll have seen that another possibility is a one-group boxplot. This might have been the context in which you first saw the boxplot, maybe at about the time you first saw the five-number summary, but I don't talk about that so much in this course because `ggplot` boxplots have both an `x` and a `y`, and it makes more sense to think about using boxplots to *compare* groups. But, you can certainly get R to make you a one-sample boxplot. What you do is to set the grouping variable to a "dummy" thing like the number 1:

```
ggplot(ben, aes(x=1, y=completed)) + geom_boxplot()
```

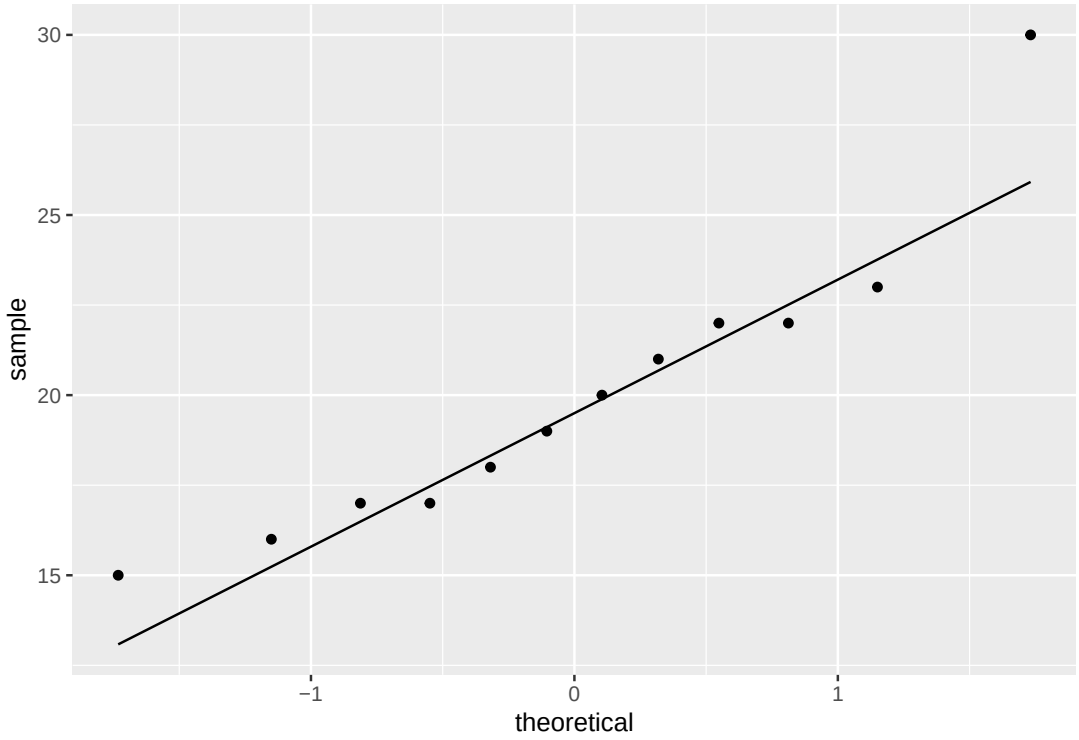


and then you ignore the x-axis.

This really shows off the outlier; it is actually *much* bigger than the other observations. It didn't show up so much on the histograms because of where the bin boundaries happened to come. On the four-bin histogram, the highest value 30 was in the 27.5–32.5 bin, and the second-highest value 23 was at the bottom of the 22.5–27.5 bin. So the highest and second-highest values looked closer together than they actually were.

If you have been reading ahead, you might also be thinking about a normal quantile plot. That is for specifically assessing normality, and here this is something that interests us, because a t-test will be doubtful if the normality fails:

```
ggplot(ben, aes(sample=completed)) + stat_qq() + stat_qq_line()
```

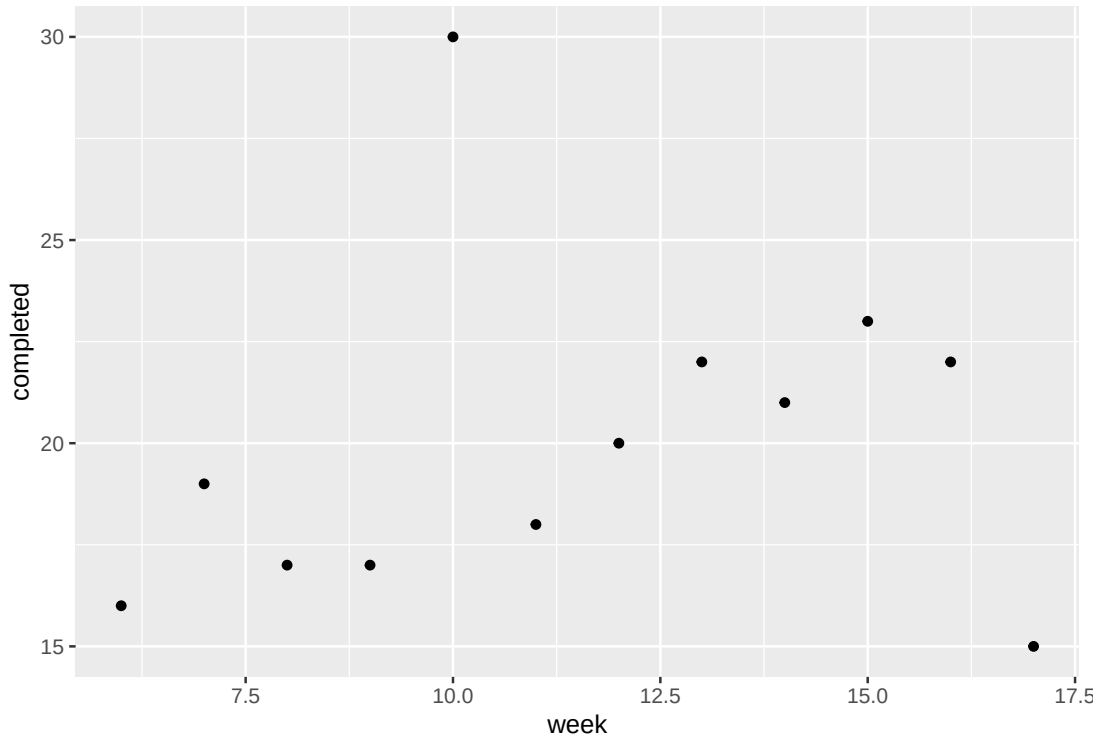


This again shows off the outlier at the high end. It is a reasonable choice of plot *here* because normality is of specific interest to us.

A note: you are absolutely not required to read ahead to future lectures. Each assignment can be done using the material in the indicated lectures only. If you want to use something from future lectures, go ahead, but make sure you are using it appropriately.

Don't be tempted to plot the number of completed passes against something like week number:

```
ggplot(ben, aes(x=week, y=completed)) + geom_point()
```



That is quite interesting (a mostly increasing trend over weeks, with the outlier performance in week 10), but it doesn't tell us what we want to know *here*: namely, is a t-test any good?

(c) Run a sign test to compare Roethlisberger's performance in 2010 with his previous average of 22.5 completions per game. What do you conclude?

Solution:

Use `smmr`, dataframe, column, null median:

```
sign_test(ben, completed, 22.5)
```

```
## $above_below
## below above
##    10     2
##
## $p_values
##   alternative    p_value
## 1       lower 0.01928711
## 2       upper 0.99682617
## 3   two-sided 0.03857422
```

I am looking for *any* change, so for me, a two-sided test is appropriate. If you think this is one-sided, make a case for your side, and then go ahead.

My P-value is 0.039, so I can reject the null hypothesis (that the median number of passes completed is 22.5) and conclude that it has changed in 2010.

(You might hypothesize that this is the result of a decrease in confidence, that he is either

throwing fewer passes, or the ones that he is throwing are harder to catch. If you know about football, you might suspect that Roethlisberger was actually passing *too much*, including in situations where he should have handing off to the running back, instead of reading the game appropriately.)

Extra: I said above that I cheated and made the null median 22.5 instead of 22. What happens if we make the null median 22?

```
sign_test(ben, completed, 22)
```

```
## $above_below
## below above
##     8     2
##
## $p_values
##   alternative   p_value
## 1       lower 0.0546875
## 2       upper 0.9892578
## 3   two-sided 0.1093750
```

For one thing, the result is no longer significant. But looking at the table of values above and below reveals something odd: there are only ten values. What happened to the other two? What happened is that two of the data values were exactly equal to 22, so they are neither above nor below. In the sign test, they are thrown away, so that we are left with 8 values below 22 and 2 above.

I didn't want to make you wonder what happened, so I made the null median 22.5.

(d) Why might you have expected your sign test to come out significant, even without looking at the P-value? Explain briefly.

Solution:

The other ingredient to the sign test is how many data values are above and below the null median. You can look at the output from `sign_test` (the first part), or count them yourself:

```
ben %>% count(completed<22.5)
```

```
## # A tibble: 2 x 2
##   `completed < 22.5`     n
##   <lgl>              <int>
## 1 FALSE                  2
## 2 TRUE                  10
```

You can put a logical condition (something that can be true or false) into `count`, or you can create a new column using `ifelse` (which I think I showed you somewhere):

```
ben %>% mutate(side = ifelse(completed<22.5, "below", "above")) %>%
  count(side)
```

```
## # A tibble: 2 x 2
##   side      n
##   <chr> <int>
## 1 above     2
## 2 below    10
```

Whichever way you do it, there seem to be a lot more values below than above, very different from a 50–50 split. Even with only 12 observations, this turns out to be enough to be significant. (If you tossed a fair coin 12 times, would you be surprised to get only 2 heads or 2 tails?)

(e) Obtain a 90% confidence interval for the median number of completed passes (over "all possible games played by 2010 Ben Roethlisberger").

Solution:

This is `ci_median`, but with `conf.level` since you are not using the default level of 95%:

```
ci_median(ben, completed, conf.level = 0.90)
```

```
## [1] 17.00244 21.99878
```

17 to 22 completed passes.

Extra: the P-value of the sign test only changes (as the null median changes) when you get to a data point; otherwise, the number of values above and below will stay the same, and the P-value will stay the same. The data values here were all whole numbers, so the limits of the confidence interval are also whole numbers (to the accuracy of the bisection), so the interval really should be rounded off.

(f) Find a 90% confidence interval for the *mean* number of passes completed, and explain briefly why it differs from the one for the median in the way that it does.

Solution:

All right, get the interval for the mean first:

```
with(ben, t.test(completed, conf.level = 0.90))
```

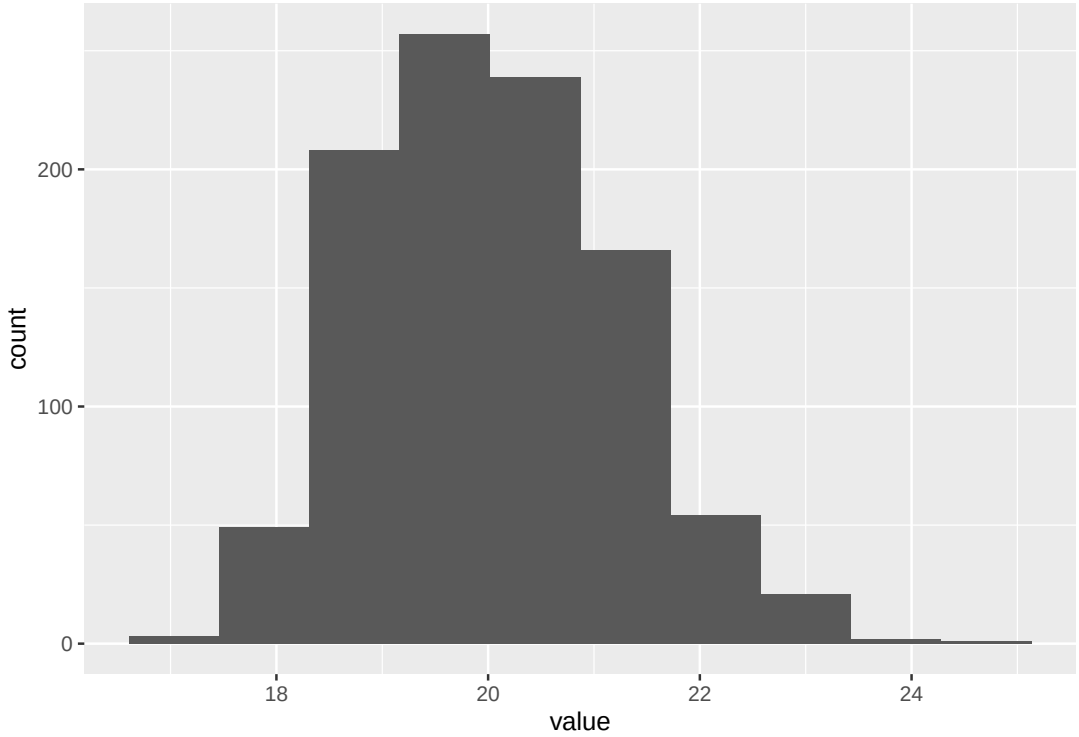
```
##
##  One Sample t-test
##
## data:  completed
## t = 17.033, df = 11, p-value = 2.971e-09
## alternative hypothesis: true mean is not equal to 0
## 90 percent confidence interval:
##  17.89124 22.10876
## sample estimates:
## mean of x
##        20
```

The 95% confidence interval for the mean goes from 17.9 to 22.1 (completions per game).

This is higher at both ends than the interval for the median, though possibly not as much as I expected. This is because the mean is made higher by the outlier (compared to the median), and so the CI procedure comes to the conclusion that the mean is higher.

Extra: this is one of those cases where the bootstrap might shed some light on the sampling distribution of the sample mean:

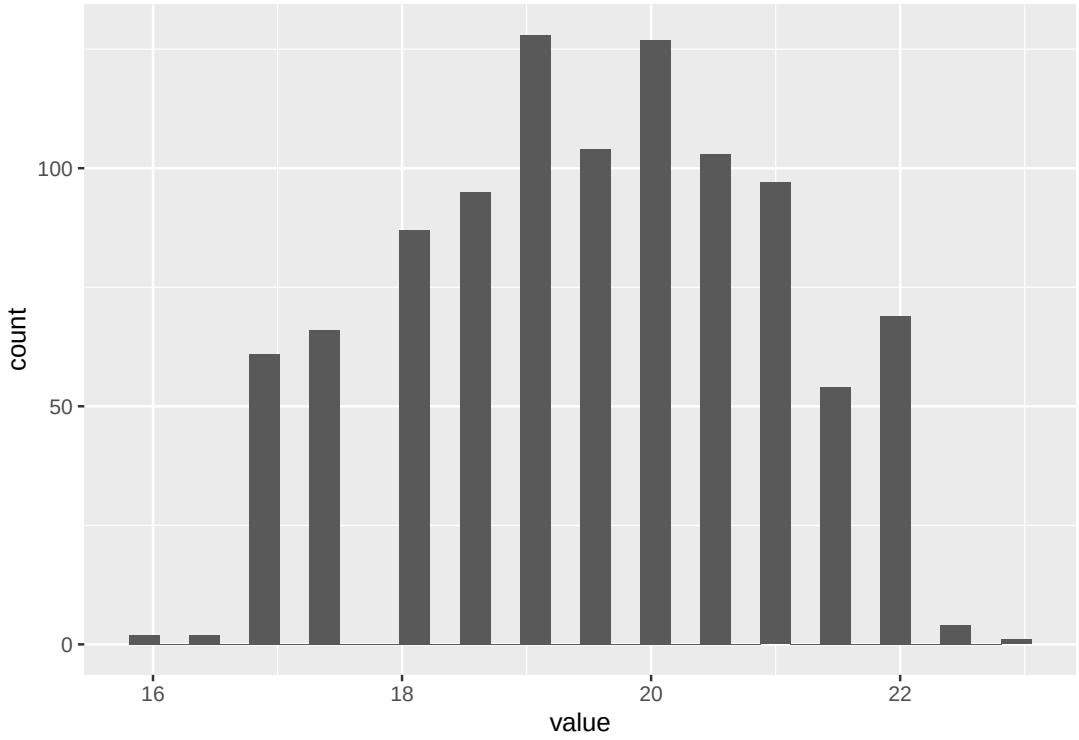
```
rerun(1000, sample(ben$completed, replace = TRUE)) %>%
  map_dbl(~mean(.)) %>%
  enframe() %>%
  ggplot(aes(x=value)) + geom_histogram(bins = 10)
```



This is a little bit skewed to the right (it goes further up from the peak than down), but not so bad, which is why the CI for the mean went up a bit higher, but only a bit, than the one for the median.

Finally, bootstrapping the median is not something you'd want to do, since the sign test doesn't depend on anything being normally-distributed. This is a good thing, since bootstrapping the sample median is weird:

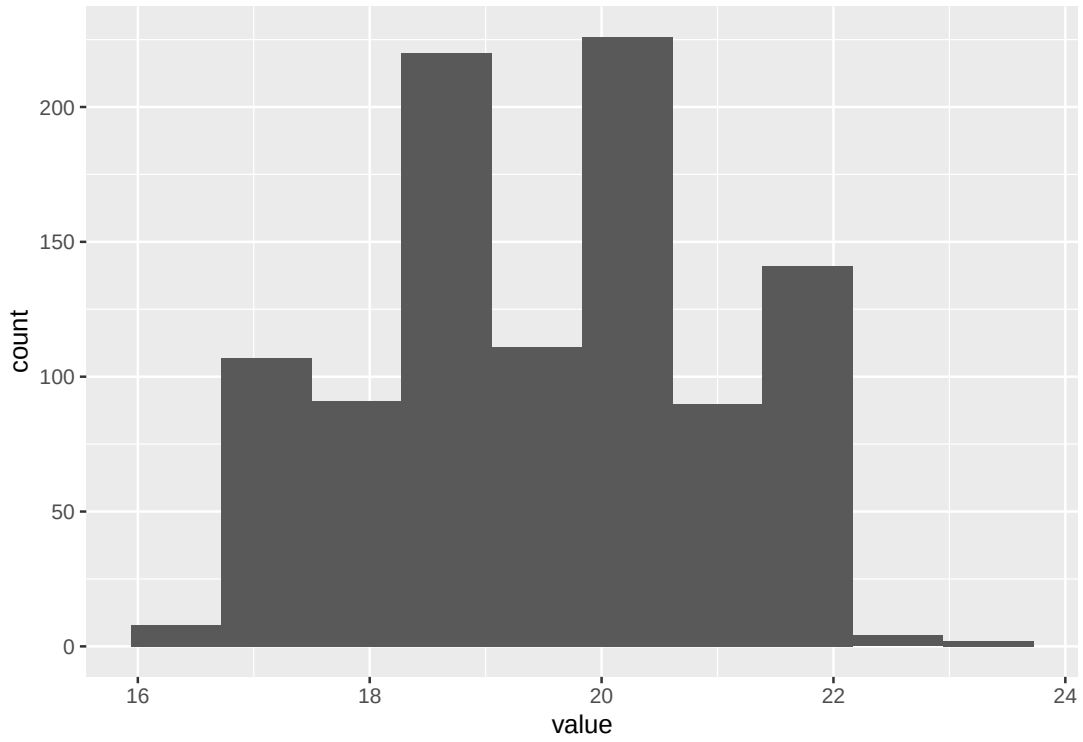
```
rerun(1000, sample(ben$completed, replace = TRUE)) %>%
  map_dbl(~median(.)) %>%
  enframe() %>%
  ggplot(aes(x=value)) + geom_histogram(bins = 30)
```



The "clumpiness" in the distribution comes about because there are not all that many different possible sample medians when you sample with replacement. For one thing, the values are all whole numbers, so the median can only be something or something and a half. Even then, the bar heights look kind of irregular.

I used a large number of bins to emphasize this, but even a more reasonable number looks strange:

```
rerun(1000, sample(ben$completed, replace = TRUE)) %>%
  map_dbl(~median(.)) %>%
  enframe() %>%
  ggplot(aes(x=value)) + geom_histogram(bins = 10)
```



A sample median of 19 or 20 is more likely than one of 19.5.

Notes

1. This is why I called my result "disappointing". I would like to reject a lot more of the time than this, but, given that the truth was not very far away from the null given the (large) population SD, I can't. See Extra 1.

2. If the power is really 0.8, the number of simulated tests that end up rejecting has a binomial distribution with n of 10000 and p of 0.80.

3. Or Reddit or Quora or whatever your favourite time-killer is.