Assignment 3

Due Tuesday February 4 at 11:59pm on Quercus

As before, the questions without solutions (here, the last one) 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 problem 8.1 of PASIAS.
2. This question is about obtaining the power by simulation for a test of a proportion (specifically, a test that the population probability of success p is 0.5).

   (a) (3 marks) The function `prop.test` takes as input an observed number of successes and a number of trials (in that order), and tests whether the population probability of success is (by default) 0.5. Suppose you toss a coin 125 times, and you obtain 80 heads. Using `prop.test`, can you conclude that the coin is biased in favour of heads?

   Solution: The null hypothesis is that the probability of a head is 0.5 (the coin is fair). The alternative is that the probability is something bigger than 0.5 (the coin is biased in favour of heads).

   What does `prop.test` say?

```r
prop.test(80, 125, alternative="greater")

##
##  1-sample proportions test with continuity correction
##
## data:  80 out of 125, null probability 0.5
## X-squared = 9.248, df = 1, p-value = 0.001179
## alternative hypothesis: true p is greater than 0.5
## 95 percent confidence interval:
##  0.5630267 1.0000000
## sample estimates:
##    p
## 0.64
```

The P-value is 0.001, so that we reject the null hypothesis and conclude that the probability of a head is something greater than 0.5 (that is, that the coin is biased in favour of heads).

(b) (3 marks) Suppose you are going to carry out 125 binomial trials. By trial and error, find the smallest number of successes that would lead you to reject $H_0 : p = 0.5$ in favour of $H_a : p > 0.5$.

Solution: The obvious way is to try different numbers of successes until you get a P-value just above 0.05 and one just below; the latter is the value you want.

We know that 80 successes rejects, so that we need a number of successes a bit less than that. 70?

```r
prop.test(70, 125, alternative="greater")

##
##  1-sample proportions test with continuity correction
##
## data:  70 out of 125, null probability 0.5
## X-squared = 1.568, df = 1, p-value = 0.1052
## alternative hypothesis: true p is greater than 0.5
## 95 percent confidence interval:
##  0.4824851 1.0000000
## sample estimates:
##    p
## 0.56
```

Does not reject, so we want something bigger. 75?

```
prop.test(75, 125, alternative="greater")
```

```
##
##  1-sample proportions test with continuity correction
##
## data:  75 out of 125, null probability 0.5
## X-squared = 4.608, df = 1, p-value = 0.01591
## alternative hypothesis: true p is greater than 0.5
## 95 percent confidence interval:
##  0.5225145 1.0000000
## sample estimates:
##   p
## 0.6
```

Rejects, so something between these.

```
prop.test(73, 125, alternative="greater")
```

```
##
##  1-sample proportions test with continuity correction
##
## data:  73 out of 125, null probability 0.5
## X-squared = 3.2, df = 1, p-value = 0.03682
## alternative hypothesis: true p is greater than 0.5
## 95 percent confidence interval:
##  0.506446 1.000000
## sample estimates:
##     p
## 0.584
```

Also rejects, so a bit smaller:

```
prop.test(72, 125, alternative="greater")
```

```
##
##  1-sample proportions test with continuity correction
##
## data:  72 out of 125, null probability 0.5
## X-squared = 2.592, df = 1, p-value = 0.0537
## alternative hypothesis: true p is greater than 0.5
## 95 percent confidence interval:
##  0.4984403 1.0000000
## sample estimates:
##     p
## 0.576
```

Does not quite reject, so now we have our answer: if we get 73 or more successes, we will reject $H_0 : p = 0.5$ in favour of $H_a : p > 0.05$. 72 won't quite do it.

A process like this is what I'm after. The exact trials and errors don't matter, as long as you are going somewhere sensible with your next guess. *Show your guesses*; at the very least, show two neighbouring values, one that gives a P-value just greater than 0.05 and one that gives a P-value just less. As is usually the way, I am interested in the *process* as much as the answer; showing me only that 73 successes will reject doesn't tell me anything about the process and does not even prove that 73 is the *smallest* number of successes that rejects.

Extra: this is a clever way of doing 70 through 80 all at once:

```
tibble(x=70:80) %>%
    mutate(test=map(x, ~prop.test(., 125, alternative="greater"))) %>%
    mutate(pvalue=map_dbl(test, "p.value"))
```

```
## # A tibble: 11 x 3
##         x test     pvalue
##     <int> <list>    <dbl>
##  1    70 <htest> 0.105
##  2    71 <htest> 0.0762
##  3    72 <htest> 0.0537
##  4    73 <htest> 0.0368
##  5    74 <htest> 0.0245
##  6    75 <htest> 0.0159
##  7    76 <htest> 0.0100
##  8    77 <htest> 0.00613
##  9    78 <htest> 0.00365
## 10    79 <htest> 0.00210
## 11    80 <htest> 0.00118
```

Once again, 73 is the first number of successes with a P-value less than 0.05.

How did that work? It uses the `map` idea to do for-each in a data frame. The first line creates a data frame with the values of `x` of 70 through 80 (inclusive). Then, for each of these numbers of successes, I run `prop.test` on that number of successes, with always 125 trials and a one-sided alternative. This gets *all* of the test output, which I don't need, so the last step is to pull out just the P-value, which is "the thing called `p.value`" in the test results. Specifically, the last line says "for each of the tests, extract the thing from it called `p.value`".

R also has a for-loop like Python:

```
for (x in 70:80) {
    test=prop.test(x, 125, alternative="greater")
    pval=test$p.value
    print(c(x, pval))
}
```

```
## [1] 70.0000000  0.1052489
## [1] 71.00000000  0.07620314
## [1] 72.00000000  0.05370232
## [1] 73.00000000  0.03681914
## [1] 74.00000000  0.02454899
## [1] 75.00000000  0.01591156
## [1] 76.00000000  0.01002233
## [1] 77.000000000  0.006133031
## [1] 78.000000000  0.003645179
## [1] 79.000000000  0.002103776
## [1] 80.000000000  0.001178764
```

I just printed the results rather than saving them, but I could have done that as well. (Saving things out of a for loop is a little more fiddly, however, since we have to create and initialize places to save them.) If you're clever, you can `break` out of the loop as soon as you hit a P-value less than 0.05.

Another way to go involves much more thinking but less coding. The P-value from this test comes from a binomial distribution with $n = 125$ and $p = 0.5$. In particular, it's the numbers of successes that have upper-tail probability just less and greater than 0.05 (or lower-tail probability either side of 0.95). This is (almost) exactly what `qbinom` does; the `q` functions are the "inverse CDF", only not exactly here because the binomial is discrete.

So:

```
qbinom(0.95, 125, 0.5)

## [1] 72
```

It's important to get the first and third inputs the right way around (the help is not very helpful in this regard): the first input is the probability whose inverse-CDF you want, and the third one is the p of the binomial distribution.

This says that the P-value will be 0.05 with "about" 72 successes. Is 72 just rejecting or just not rejecting? This requires some care: probability of 72 or more is one minus the probability of *71* or less:

```
1-pbinom(71, 125, 0.5) # 72 or more

## [1] 0.05351571

1-pbinom(72, 125, 0.5) # 73 or more

## [1] 0.03660716
```

73 is the first one that rejects.

(c) (2 marks) The function `rbinom` takes random samples from a binomial distribution. Find out how it works. The first three inputs are what concern you. Obtain three random binomials with $n = 4$ and $p = 0.3$.

Solution: One way to find out is to go down to the Console window and type `?rbinom`. This brings up the help for the binomial distribution. `rbinom` is the bottom one, with inputs `n, size, prob`. Careful here: `n` is not what you think it is: it is the number of observations, that is, the number of random binomials, to generate. The second input `size`, is the n (number of trials) for each binomial, and the third input, `prob`, is the probability of success.

All right, now that we have that, the inputs have to be 3, 4 and 0.3 *in that order*:

```
rbinom(3, 4, 0.3)

## [1] 1 0 1
```

You should get three small numbers (4 or less, and probably 0 or 1). What you get doesn't matter if you have the code right.

(d) (4 marks) Use simulation to estimate the power of the z-test for a proportion to reject $H_0 : p = 0.5$ in favour of $H_a : p > 0.5$, when in fact $p = 0.6$. For full marks, do this without using `prop.test`, `rerun` or any `map`; in the previous parts, you worked out what numbers of successes would lead to rejection and what numbers would not, and how to generate any number of random binomials from distributions with given n and p.

Solution: Here's the best procedure, given what we have done so far:

- Generate lots of random samples from a binomial distribution with $n = 125, p = 0.6$
- For each of these, if the number of successes is 73 or greater, then reject, otherwise don't reject.
- Count up the rejections out of the total.

In code, this, with 1000 standing in for "lots":

```
tibble(x=rbinom(1000, 125, 0.6)) %>%
  mutate(decision=ifelse(x>=73, "reject", "do not reject")) %>%
  count(decision)
```

```
## # A tibble: 2 x 2
##   decision          n
##   <chr>         <int>
## 1 do not reject   326
## 2 reject          674
```

The estimated power is something like 68%, depending on the particular random numbers you happen to get. This also works:

```
tibble(x=rbinom(1000, 125, 0.6)) %>%
  mutate(reject=(x>=73)) %>%
  count(reject)
```

```
## # A tibble: 2 x 2
##   reject     n
##   <lgl>  <int>
## 1 FALSE    298
## 2 TRUE     702
```

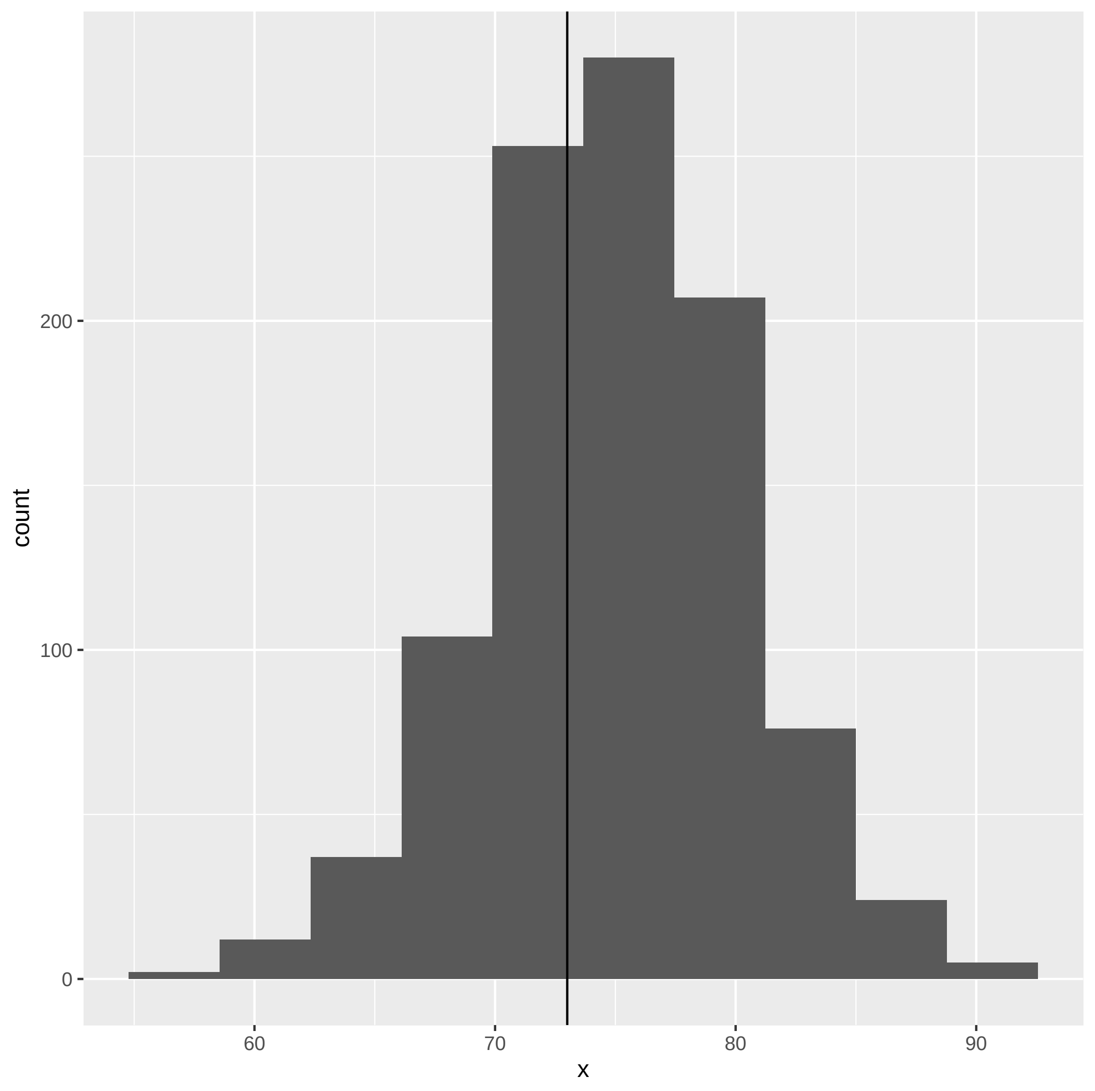
(any difference from the first way is due to randomness). Or, simpler still:

```
tibble(x=rbinom(1000, 125, 0.6)) %>%
  count(x>=73)
```

```
## # A tibble: 2 x 2
##   `x >= 73`     n
##   <lgl>     <int>
## 1 FALSE       331
## 2 TRUE        669
```

For extra understanding, we can repeat any of these but plot it instead:

```
tibble(x=rbinom(1000, 125, 0.6)) %>%
  ggplot(aes(x=x)) + geom_histogram(bins=10) +
  geom_vline(xintercept=73)
```



This shows what numbers of successes you might get when $p = 0.6$ in 125 trials, somewhere between 60 and 90, but a bit more than half the time you'll get 73 successes or more (the vertical line) and end up correctly rejecting $H_0 : p = 0.5$.

This is a lot easier coding than what we had in class, and if you get what power is doing, a lot easier to understand as well: "generate a bunch of samples from the truth, and find out how many of them reject the null".

If you have learned R elsewhere, you may have run into a for-loop, which is another way to run the simulation:

```r
count <- 0
n_sim <- 1000
for (i in 1:n_sim) {
    x=rbinom(1, 125, 0.6)
    if (x>=73) {
        count <- count+1
    }
}
count/n_sim
```

```
## [1] 0.697
```

This generates the random binomials one at a time, tests each one to see if it would lead to a rejection, and keeps track of how many rejections you've seen so far. It is inefficient in R terms, because R lets you keep track of vectors of things all at one, and so you don't need the loop, but it works, so it is good. (This is how a Python programmer would do it.)

If you're going to religiously follow what I did in class, you have to do something like this, which is a lot more work:

```r
rerun(1000, rbinom(1, 125, 0.6)) %>%
  map( ~prop.test(., 125, 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            309
## 2 TRUE             691
```

This, if you get it right, is most of the marks but not full marks, because you have shown ability to adapt other code, but not understanding of what it is doing.