The sign test

Packages

```
library(tidyverse)
library(smmr)
```

`smmr` is new. See later how to install it.

Duality between confidence intervals and hypothesis tests

- Tests and CIs really do the same thing, if you look at them the right way. They are both telling you something about a parameter, and they use same things about data.
- To illustrate, some data (two groups):

```
my_url <- "http://ritsokiguess.site/datafiles/duality.txt"
twogroups <- read_delim(my_url," ")
```

The data

```
twogroups
```

```
# A tibble: 15 x 2
       y group
   <dbl> <dbl>
 1    10     1
 2    11     1
 3    11     1
 4    13     1
 5    13     1
 6    14     1
 7    14     1
 8    15     1
 9    16     1
10    13     2
11    13     2
12    14     2
```

95% CI (default)

for difference in means, group 1 minus group 2:

```
t.test(y ~ group, data = twogroups)
```

```

    Welch Two Sample t-test

data:  y by group
t = -2.0937, df = 8.7104, p-value = 0.0668
alternative hypothesis: true difference in means between grou
95 percent confidence interval:
 -5.5625675  0.2292342
sample estimates:
mean in group 1 mean in group 2
       13.00000        15.66667
```

90% CI

```r
t.test(y ~ group, data = twogroups, conf.level = 0.90)
```

```

	Welch Two Sample t-test

data:  y by group
t = -2.0937, df = 8.7104, p-value = 0.0668
alternative hypothesis: true difference in means between grou
90 percent confidence interval:
 -5.010308 -0.323025
sample estimates:
mean in group 1 mean in group 2
       13.00000        15.66667
```

Hypothesis test

Null is that difference in means is zero:

```
t.test(y ~ group, mu=0, data = twogroups)
```

```

	Welch Two Sample t-test

data:  y by group
t = -2.0937, df = 8.7104, p-value = 0.0668
alternative hypothesis: true difference in means between grou
95 percent confidence interval:
 -5.5625675  0.2292342
sample estimates:
mean in group 1 mean in group 2 
       13.00000        15.66667 
```

Comparing results

Recall null here is $H_0 : \mu_1 - \mu_2 = 0$. P-value 0.0668.

- 95% CI from -5.6 to 0.2, contains 0.
- 90% CI from -5.0 to -0.3, does not contain 0.
- At $\alpha = 0.05$, would not reject H_0 since P-value > 0.05.
- At $\alpha = 0.10$, *would* reject H_0 since P-value < 0.10.

Test and CI

Not just coincidence. Let $C = 100(1-\alpha)$, so C% gives corresponding CI to level-α test. Then following always true. (Symbol $\iff$ means "if and only if".)

Test decision		Confidence interval
Reject H_0 at level α	$\iff$	$C\%$ CI does not contain H_0 value
Do not reject H_0 at level α	$\iff$	$C\%$ CI contains H_0 value

Idea: "Plausible" parameter value inside CI, not rejected; "Implausible" parameter value outside CI, rejected.

The value of this

- If you have a test procedure but no corresponding CI:
- you make a CI by including all the parameter values that would not be rejected by your test.
- Use:
  - $\alpha = 0.01$ for a 99% CI,
  - $\alpha = 0.05$ for a 95% CI,
  - $\alpha = 0.10$ for a 90% CI, and so on.

Testing for non-normal data

- The IRS ("Internal Revenue Service") is the US authority that deals with taxes (like Revenue Canada).
- One of their forms is supposed to take no more than 160 minutes to complete. A citizen's organization claims that it takes people longer than that on average.
- Sample of 30 people; time to complete form recorded.
- Read in data, and do t-test of $H_0 : \mu = 160$ vs. $H_a : \mu > 160$.
- For reading in, there is only one column, so can pretend it is delimited by anything.

Read in data

```
my_url <- "http://ritsokiguess.site/datafiles/irs.txt"
irs <- read_csv(my_url)
irs
```

```
# A tibble: 30 x 1
    Time
   <dbl>
 1    91
 2    64
 3   243
 4   167
 5   123
 6    65
 7    71
 8   204
 9   110
10   178
```

Test whether mean is 160 or greater

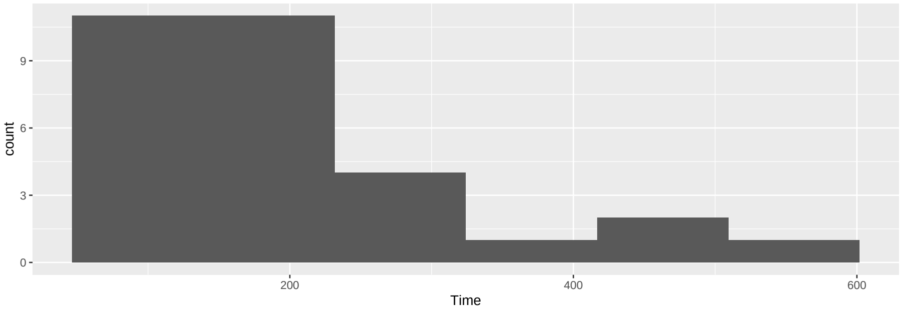
```
with(irs, t.test(Time, mu = 160,
                 alternative = "greater"))
```

```

    One Sample t-test

data:  Time
t = 1.8244, df = 29, p-value = 0.03921
alternative hypothesis: true mean is greater than 160
95 percent confidence interval:
 162.8305      Inf
sample estimates:
mean of x
 201.2333
```

Reject null; mean (for all people to complete form) greater than 160.

But, look at a graph

```
ggplot(irs, aes(x = Time)) + geom_histogram(bins = 6)
```



Comments

- Skewed to right.
- Should look at *median*, not mean.

The sign test

- But how to test whether the median is greater than 160?
- Idea: if the median really is 160 (H_0 true), the sampled values from the population are equally likely to be above or below 160.
- If the population median is greater than 160, there will be a lot of sample values greater than 160, not so many less. Idea: test statistic is number of sample values greater than hypothesized median.

Getting a P-value for sign test 1/3

- How to decide whether "unusually many" sample values are greater than 160? Need a sampling distribution.
- If H_0 true, pop. median is 160, then each sample value independently equally likely to be above or below 160.
- So number of observed values above 160 has binomial distribution with $n = 30$ (number of data values) and $p = 0.5$ (160 is hypothesized to be *median*).

Getting P-value for sign test 2/3

- Count values above/below 160:

```
irs %>% count(Time > 160)
```

```
# A tibble: 2 x 2
  `Time > 160`     n
  <lgl>        <int>
1 FALSE           13
2 TRUE            17
```

- 17 above, 13 below. How unusual is that? Need a *binomial table*.

Getting P-value for sign test 3/3

- R function `dbinom` gives the probability of eg. exactly 17 successes in a binomial with $n = 30$ and $p = 0.5$:

```
dbinom(17, 30, 0.5)
```

```
[1] 0.1115351
```

- but we want probability of 17 *or more*, so get all of those, find probability of each, and add them up:

```
tibble(x=17:30) %>%
  mutate(prob=dbinom(x, 30, 0.5)) %>%
  summarize(total=sum(prob))
```

```
# A tibble: 1 x 1
  total
  <dbl>
1 0.292
```

Using my package smmr

- I wrote a package `smmr` to do the sign test (and some other things). Installation is a bit fiddly:
  - Install devtools (once) with

```
install.packages("devtools")
```

- then install `smmr` using `devtools` (once):

```
library(devtools)
install_github("nxskok/smmr")
```

- Then load it:

```
library(smmr)
```

smmr for sign test

- smmr's function `sign_test` needs three inputs: a data frame, a column and a null median:

```
sign_test(irs, Time, 160)
```

```
$above_below
below above
   13    17

$p_values
  alternative   p_value
1       lower 0.8192027
2       upper 0.2923324
3   two-sided 0.5846647
```

Comments (1/3)

- Testing whether population median *greater than* 160, so want *upper-tail* P-value 0.2923. Same as before.
- Also get table of values above and below; this too as we got.

Comments (2/3)

- P-values are:

Test	P-value
t	0.0392
Sign	0.2923

- These are very different: we reject a mean of 160 (in favour of the mean being bigger), but clearly *fail* to reject a median of 160 in favour of a bigger one.
- Why is that? Obtain mean and median:

```
irs %>% summarize(mean_time = mean(Time),
                  median_time = median(Time))
```

```
# A tibble: 1 x 2
  mean_time median_time
      <dbl>       <dbl>
```

Comments (3/3)

- The mean is pulled a long way up by the right skew, and is a fair bit bigger than 160.
- The median is quite close to 160.
- We ought to be trusting the sign test and not the t-test here (median and not mean), and therefore there is no evidence that the "typical" time to complete the form is longer than 160 minutes.
- Having said that, there are clearly some people who take a lot longer than 160 minutes to complete the form, and the IRS could focus on simplifying its form for these people.
- In this example, looking at any kind of average is not really helpful; a better question might be "do an unacceptably large fraction of people take longer than (say) 300 minutes to complete the form?": that is, thinking about worst-case rather than average-case.

Confidence interval for the median

- The sign test does not naturally come with a confidence interval for the median.
- So we use the "duality" between test and confidence interval to say: the (95%) confidence interval for the median contains exactly those values of the null median that would not be rejected by the two-sided sign test (at $\alpha = 0.05$).

For our data

- The procedure is to try some values for the null median and see which ones are inside and which outside our CI.
- smmr has pval_sign that gets just the 2-sided P-value:

```
pval_sign(160, irs, Time)
```

```
[1] 0.5846647
```

- Try a couple of null medians:

```
pval_sign(200, irs, Time)
```

```
[1] 0.3615946
```

```
pval_sign(300, irs, Time)
```

```
[1] 0.001430906
```

- So 200 inside the 95% CI and 300 outside.

Doing a whole bunch

- Choose our null medians first:

```
(d <- tibble(null_median=seq(100,300,20)))
```

```
# A tibble: 11 x 1
   null_median
         <dbl>
 1         100
 2         120
 3         140
 4         160
 5         180
 6         200
 7         220
 8         240
 9         260
10         280
11         300
```

... and then

"for each null median, run the function `pval_sign` for that null median and get the P-value":

```
d %>% rowwise() %>%
  mutate(p_value = pval_sign(null_median, irs, Time))
```

```
# A tibble: 11 x 2
# Rowwise:
   null_median  p_value
         <dbl>    <dbl>
 1         100 0.000325
 2         120 0.0987
 3         140 0.200
 4         160 0.585
 5         180 0.856
 6         200 0.362
 7         220 0.0428
 8         240 0.0161
```

Make it easier for ourselves

```
d %>% rowwise() %>%
  mutate(p_value = pval_sign(null_median, irs, Time)) %>%
  mutate(in_out = ifelse(p_value > 0.05, "inside", "outside"))
```

```
# A tibble: 11 x 3
# Rowwise:
   null_median  p_value in_out
         <dbl>    <dbl> <chr>
 1         100 0.000325 outside
 2         120 0.0987   inside
 3         140 0.200    inside
 4         160 0.585    inside
 5         180 0.856    inside
 6         200 0.362    inside
 7         220 0.0428   outside
 8         240 0.0161   outside
 9         260 0.00522  outside
```

confidence interval for median?

- 95% CI to this accuracy from 120 to 200.
- Can get it more accurately by looking more closely in intervals from 100 to 120, and from 200 to 220.

A more efficient way: bisection

- Know that top end of CI between 200 and 220:

```
lo <- 200
hi <- 220
```

- Try the value halfway between: is it inside or outside?

```
try <- (lo + hi) / 2
try
```

```
[1] 210
```

```
pval_sign(try,irs,Time)
```

```
[1] 0.09873715
```

- Inside, so upper end is between 210 and 220. Repeat (over):

… bisection continued

```
lo <- try
try <- (lo + hi) / 2
try
```

```
[1] 215
```

```
pval_sign(try, irs, Time)
```

```
[1] 0.06142835
```

- 215 is inside too, so upper end between 215 and 220.
- Continue until have as accurate a result as you want.

Bisection automatically

- A loop, but not a `for` since we don't know how many times we're going around. Keep going `while` a condition is true:

```
lo = 200
hi = 220
while (hi - lo > 1) {
  try = (hi + lo) / 2
  ptry = pval_sign(try, irs, Time)
  print(c(try, ptry))
  if (ptry <= 0.05)
    hi = try
  else
    lo = try
}
```

The output from this loop

```
[1] 210.00000000   0.09873715
[1] 215.00000000   0.06142835
[1] 217.50000000   0.04277395
[1] 216.25000000   0.04277395
[1] 215.62500000   0.04277395
```

- 215 inside, 215.625 outside. Upper end of interval to this accuracy is 215.

Using smmr

- `smmr` has function `ci_median` that does this (by default 95% CI):

```
ci_median(irs, Time)
```

```
[1] 119.0065 214.9955
```

- Uses a more accurate bisection than we did.
- Or get, say, 90% CI for median:

```
ci_median(irs, Time, conf.level=0.90)
```

```
[1] 123.0031 208.9960
```

- 90% CI is shorter, as it should be.

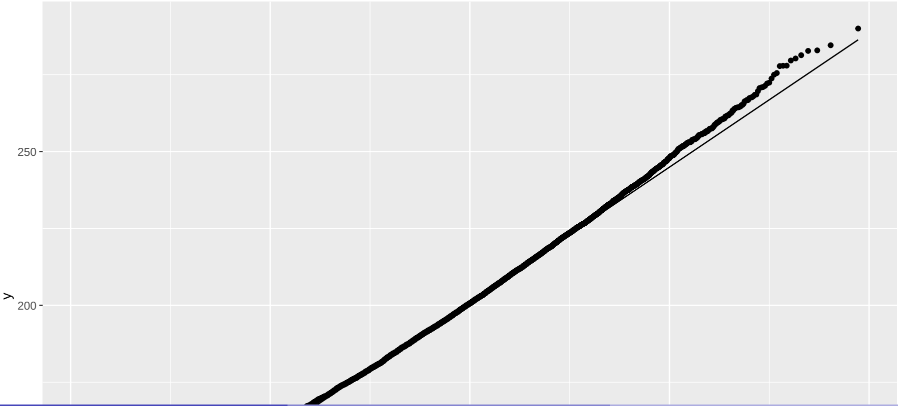
Bootstrap

- but, was the sample size (30) big enough to overcome the skewness?
- Bootstrap, again:

```
tibble(sim = 1:1000) %>%
  rowwise() %>%
  mutate(my_sample = list(sample(irs$Time, replace = TRUE))) %
  mutate(my_mean = mean(my_sample)) %>%
  ggplot(aes(x=my_mean)) + geom_histogram(bins=10) -> g
```

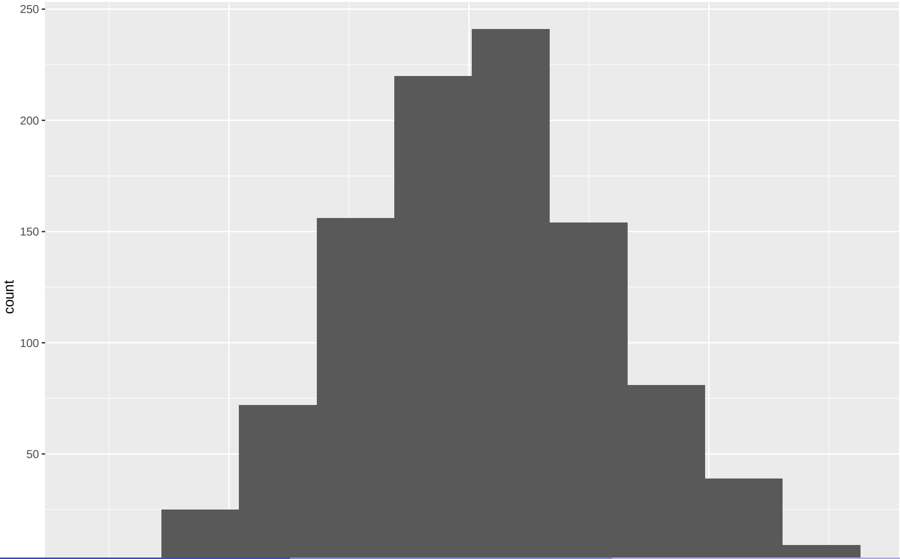
With normal quantile plot (see after that section)

```
tibble(sim = 1:10000) %>%
  rowwise() %>%
  mutate(my_sample = list(sample(irs$Time, replace = TRUE))) %
  mutate(my_mean = mean(my_sample)) %>%
  ggplot(aes(sample = my_mean)) + stat_qq() + stat_qq_line()
```



The sampling distribution

```
g
```



Comments

- A little skewed to right, but not nearly as much as I was expecting.
- The t-test for the mean might actually be OK for these data, *if the mean is what you want*.
- In actual data, mean and median very different; we chose to make inference about the median.
- Thus for us it was right to use the sign test.