LAB 04

STAT 131A

Fed 14, 2022

Welcome to the Lab 4! In this lab, you will:

1) Illustrate the difference between a T-distribution and the standard normal distribution
2) Illustrate how functions work in `R`
3) Simulation examples when T-test falls

T distribution versus the normal distribution

The T distribution has a mean of 0 and a variance of $n/(n-2)$. It is parameterized by just one parameter, the degrees of freedom. The degrees of freedom for practical purposes is equal to the sample size minus 1 for a one sample T-test. For a two sample T-test, as done above, the degrees of freedom is calculated through a formula (that you don't need to know).

The T distribution is similar in shape to the standard normal distribution, but the relative difference in the tail regions of the distributions is significant. This implies that for hypothesis tests, the p-value we find will be quite different if we use the wrong distribution as our sampling distribution for the test statistic.

The following code finds the tail probability for a T-score (or t-statistic / t-value) of -1, with 30 degrees of freedom.

```r
pt(-1, df = 30)
```

```
## [1] 0.1626543
```

We can see that this different from that of a Z-score (standard normal distribution) of -1.

```r
pnorm(-1)
```

```
## [1] 0.1586553
```

The difference is larger when we change the degrees of freedom to 10.

```r
pt(-1, df = 10)
```

```
## [1] 0.1704466
```

Exercise 1

(a) Find the tail probability of a t-score of -2 with 30 degrees of freedom. Divide this by the tail probability of a z-score of -2.

```r
# Insert code here
# Save final answer (ratio) as `e1a`
e1a = NULL
```

(b) Repeat part (a) but for 10 degrees of freedom.

```r
# Insert code here
# Save final answer (ratio) as `e1b`
e1b = NULL
```

It is common to run a hypothesis test with a significance level of 5 percent, or 0.05. This corresponds to rejecting the null hypothesis if we are more than 2 standard errors from the mean, or a z-score of less than -2 or more than 2. However, if we do not know the true standard error, and we use the sample standard deviation to estimate it, then we must use the T-distribution. You can see from above that in this case our true p-value is much greater than we would otherwise think it is under the standard normal distribution. This is the effect of the uncertainty that comes from not knowing the true population standard deviation.

`apply`, `sapply`, and `function`

Previously, we introduced `replicate` to repeat a expression several times. Now what if we want to run a function several times, but change the argument every time we run it? For example, to calculate the square root of integers from 1 to 100. Though `for` loops would certainly work, there is a much faster and simpler way in `R`. Consider the two functions `apply` and `sapply`.

`apply` traverses row- or column-wise and applies a function to each row (or column). It is usually used on matrix and data frames. Depending on the function, it returns a vector, array, or list of values.

`sapply` traverses every element in an array or a list and applies a function on each element.

Let us look at the problem of calculating the square root of integers plus the integer value itself from 1 to 100. To implement with for loops:

```r
sqroots = c()
for (i in 1:100) {
  sqroots = c(sqroots, i + sqrt(i))
}
```

And it is equivalent to the following with `sapply`:

```r
sqroots <- sapply(1:100, function(x) {
  x + sqrt(x)
})
```

`sapply` usually results in a shorter run time and easier code implementation.

Now we create a matrix with 100 rows and 10 columns with each entries being a random number from $N(0, 1)$.

```r
mat <- matrix(rnorm(1000), 100)
```

To obtain the maximum number of each row with `apply`:

```r
row.max <- apply(mat, 1, max)
```

To obtain the sum of the third the the fifth element each row with `apply`:

```r
row.sum35 <- apply(mat, 1, function(x) x[3] + x[5])
```

To obtain the maximum number of each column with `apply`:

```r
col.max <- apply(mat, 2, max)
```

As you can see above, a function is different from an expression in that a function can take inputs and give an output specific to that input. Functions can also be given names, and saved in the environment. What a function executes can be wrapped in curly braces if there are multiple lines. In the function declaration, the parameters of the function must be named and listed. In the body of the function, those named parameters can be called on and operated upon. The value returned by a function does not have to be coded explicitly. They return the last output if `return()` is not invoked.

```r
lab131 <- function(x, yy){
  x-yy
  yy-x}

lab131(5,3)
```

```
## [1] -2
```

```r
lab131 <- function(x, yy){
  return(x-yy)
  yy-x}

lab131(5,3)
```

```
## [1] 2
```

```r
lab131 <- function(x, yy){

  }

lab131(5,3)
```

```
## NULL
```

Exercise 2

Write a function called ‘stat131’ with parameters called `st` and `lb` (no quotes) that adds the square root of `st` to `lb` and returns the resulting sum.

```r
# Insert answer here
stat131 <- function(x, y) {}
```

Simulation: when will the t-test fail

Following four functions generate four types of sample (You can ignore the details concerning how the samples are generated.)

- **generate.sample1**: Small sample size (5) and normal distributed samples.
- **generate.sample2**: Small sample size (5) and non-normal distributed samples.
- **generate.sample3**: Large sample size (100) and non-normal distributed samples.
- **generate.sample4**: Large sample size (100), normal distirbuted but sample not independent.

The Gamma distribution with parameters shape = a and scale = s has density

$$f(x) = \frac{x^{(a-1)}e^{-(x/s)}}{s^a\Gamma(a)}$$

```r
generate.sample1 <- function(){
  # generate 5 samples from N(0, 1)
  sample <- rnorm(n = 5)
  return(sample)
}

generate.sample2 <- function(){
  # generate 5 samples from Gamma distribution
  sample <- rgamma(n = 5, shape = 1.5, rate = 2)
  return(sample)
}

generate.sample3 <- function(){
  # generate 100 samples from Cauchy distribution
  sample <- rcauchy(100, 0, 1)
  return(sample)
}

generate.sample4 <- function(){
  # generate 100 samples from an ARIMA model
  sample <- arima.sim(n = 100, list(ar = c(1, -0.5), ma = c(0.3)))
  return(sample)
}
```

If you run the function, such as `generate.sample1`, it will generate the corresponding samples.

```r
samples <- generate.sample1()
samples
```

```
## [1] -1.7185537 -0.5354118 -0.7832818  1.1609456  0.3972423
```

Exercise 3

Please complete the function `simulation`. It accept the name of the above sample generation functions and return the estimated type I error. Please follow the following steps for completing the function `simulation`:

1. generate two samples from the function `FUN`.

2. calculate the p-value using t-test.
3. repeat 1 and 2 for N times.
4. calculate the chance of rejecting the null hypothesis $H_0 : \mu_1 = \mu_2$ and alternative $H_1 : \mu_1 \neq \mu_2$ when the significant level is set to 0.05. (that is, the estimated Type I Error) HINT: The chance of rejecting the null can be estimated by ((The number of rejections) / N) = ((The number of p values less or equal than 0.05) / N).

```r
simulation <- function(FUN, N = 10000){
  # FUN: function name for sample generation
  # N: number of repetitions


}
```

After complete your function above, you can run the following code:

```r
set.seed(123456) # please do not modify the seed
simulation(generate.sample1)
```

```
## NULL
```

```r
set.seed(123456) # please do not modify the seed
simulation(generate.sample2)
```

```
## NULL
```

```r
set.seed(123456) # please do not modify the seed
simulation(generate.sample3)
```

```
## NULL
```

```r
set.seed(123456) # please do not modify the seed
simulation(generate.sample4)
```

```
## NULL
```

Since the null hypothesis is true when we generate the data, the estimated Type I error should be close to 0.05 as we increase the number of repetitions. However, you may observe that t-test fails except for the first sample.

From the simulation, which test do you think is better for the Craigslist dataset? why?

```
Write your non-coding answer here.
```