Lab 3: Randomisation, Loops, Normal Inference

Complete all of the following questions, adding your inputs as code chunks (enclose within triple accent marks) within Rmarkdown.

The exercises are not marked and will not be factored into your course grade, but it is important to complete them to make sure you have the skills to answer assessment questions. You may consult any resource, including other students and the instructor. Please Knit this document to a PDF and upload your work via Canvas at the end of the session. Solutions will be posted for you to check your own answers.

Random variates

1. Create a vector containing a large number of values (at least 10^5) drawn from the standard normal distribution. Verify that the standard deviation is close to the theoretical value.

```
z = rnorm(100000)
sd(z)
```

```
## [1] 0.9999112
```

2. Numerically estimate the mean and standard deviation of a uniform distribution, where the PDF is equal to 1 between $0 < x < 1$ and equal to 0 everywhere else.

```
u = runif(100000, min=0, max=1)
mean(u)
```

```
## [1] 0.5016891
```

```
sd(u)
```

```
## [1] 0.2880655
```

```
# note: analytic SD is sqrt(1/12)
```

3. Numerically estimate the mean and standard deviation for the distribution formed by rolling a single six-sided (fair) die.

```
die = sample(1:6,100000,replace=TRUE)
mean(die)
```

```
## [1] 3.50012
```

```
sd(die)
```

```
## [1] 1.705142
```

```
# note: analytic SD is sqrt(35/12)
```

4. Numerically demonstrate (with a few examples) that the variance of the distribution formed by summing data sampled from two separate, independent normal distributions is equal to the sum of the component variances (for large N).

```
sim1 = rnorm(100000,sd=1)+rnorm(100000,sd=1)
sd(sim1)
```

```
## [1] 1.414044
```

```
sqrt(2) # sqrt(1^2+1^2)
```

```
## [1] 1.414214
```

```
sim2 = rnorm(100000,sd=3)+rnorm(100000,sd=4)
sd(sim2)
```

```
## [1] 5.021692
```

```
sqrt(3^2+4^2)
```

```
## [1] 5
```

Loops

5. Store your name in a character (string) variable. Then, write a loop that prints out all the letters one by one, using `substr`.

```
name = "Daniel Perley"
for (i in 1:nchar(name)) print(substr(name,i,i))
```

```
## [1] "D"
## [1] "a"
## [1] "n"
## [1] "i"
## [1] "e"
## [1] "l"
## [1] " "
## [1] "P"
## [1] "e"
## [1] "r"
## [1] "l"
## [1] "e"
## [1] "y"
```

6. Write a loop that prints out every other letter in your name, backwards, starting with the final letter.

```
for (i in seq(nchar(name),1,-2)) print(substr(name,i,i))
```

```
## [1] "y"
## [1] "l"
## [1] "e"
## [1] " "
## [1] "e"
## [1] "n"
## [1] "D"
```

7. Write a loop that does the following: starting with the variable n = 1, each the time the loop runs, the value n is doubled. Keep looping as long as n remains less than 1 trillion (10^{12}). Once n exceeds a trillion, stop and print out n. (Use a while loop.)

```
n = 1
while(n < 1e12) { n = n*2;}
n
```

```
## [1] 1.099512e+12
```

8. Create a 4x6 matrix m, initially containing all zeroes. Then, use a nested loop (a loop inside a loop) to populate each element (x, y) of the matrix with a value equal to x*y. Then, print out the matrix.

```
m = matrix(0,4,6)
for(x in 1:4) {
    for(y in 1:6) {
        m[x,y] = x*y
    }
}
m
```

```
##      [,1] [,2] [,3] [,4] [,5] [,6]
## [1,]    1    2    3    4    5    6
## [2,]    2    4    6    8   10   12
## [3,]    3    6    9   12   15   18
## [4,]    4    8   12   16   20   24
```

Simulations

9. Write a loop that runs 10 times. Each time the loop iterates, it should simulate rolling three six-sided die and print out their sum. Run the loop and look at the output.

```
for(i in 1:10) { print( sum(sample(1:6,3,replace=TRUE) ))}
```

```
## [1] 10
## [1] 11
## [1] 10
## [1] 10
## [1] 11
## [1] 11
## [1] 11
## [1] 8
## [1] 12
## [1] 12
```

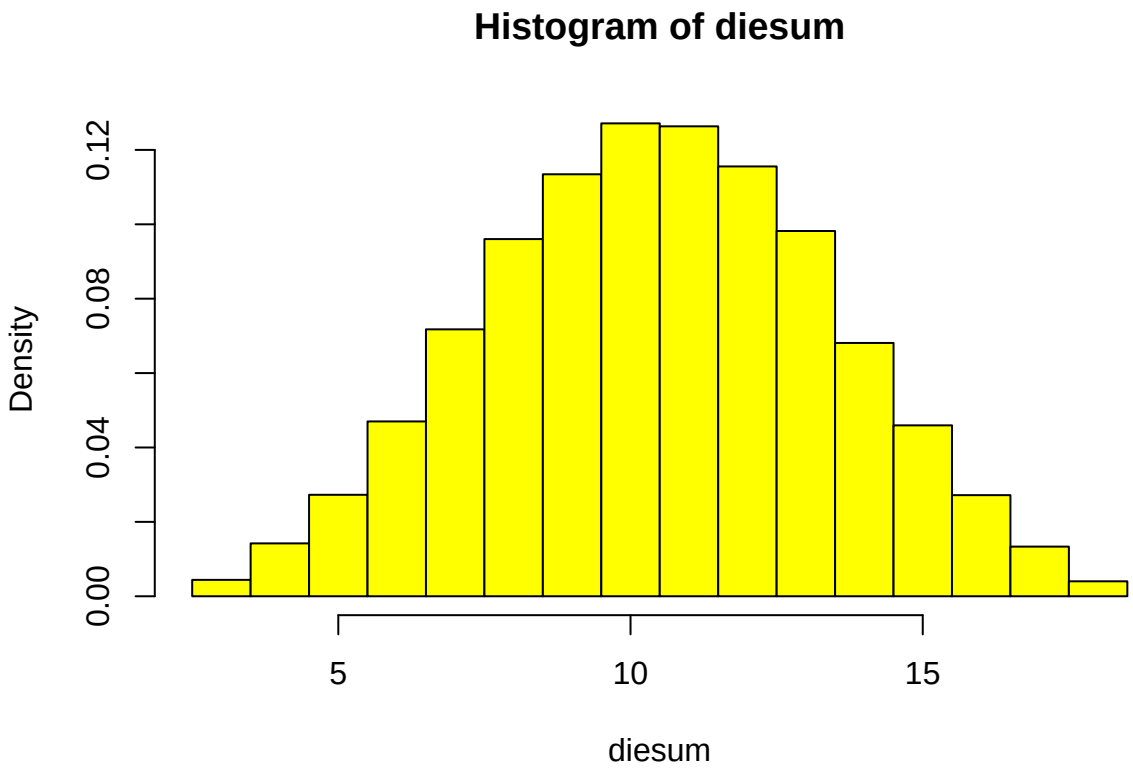
10. Write a loop similar to the one above, but instead of printing out the sum, store each result in an element of a vector. To do this:
    a. Create a vector "diesum" that is 10 elements long, initially all zeroes.
    b. Use the loop you wrote from #9 above, but store the summed result in the i^{th} element of diesum instead of printing it out.
    c. After the result finishes, print out the vector.

```
diesum = rep(0,10)
for(i in 1:10) { diesum[i] = sum(sample(1:6,3,replace=TRUE) ) }
diesum
```

```
##  [1] 10 12  5  9 10 10 11  6 15 13
```

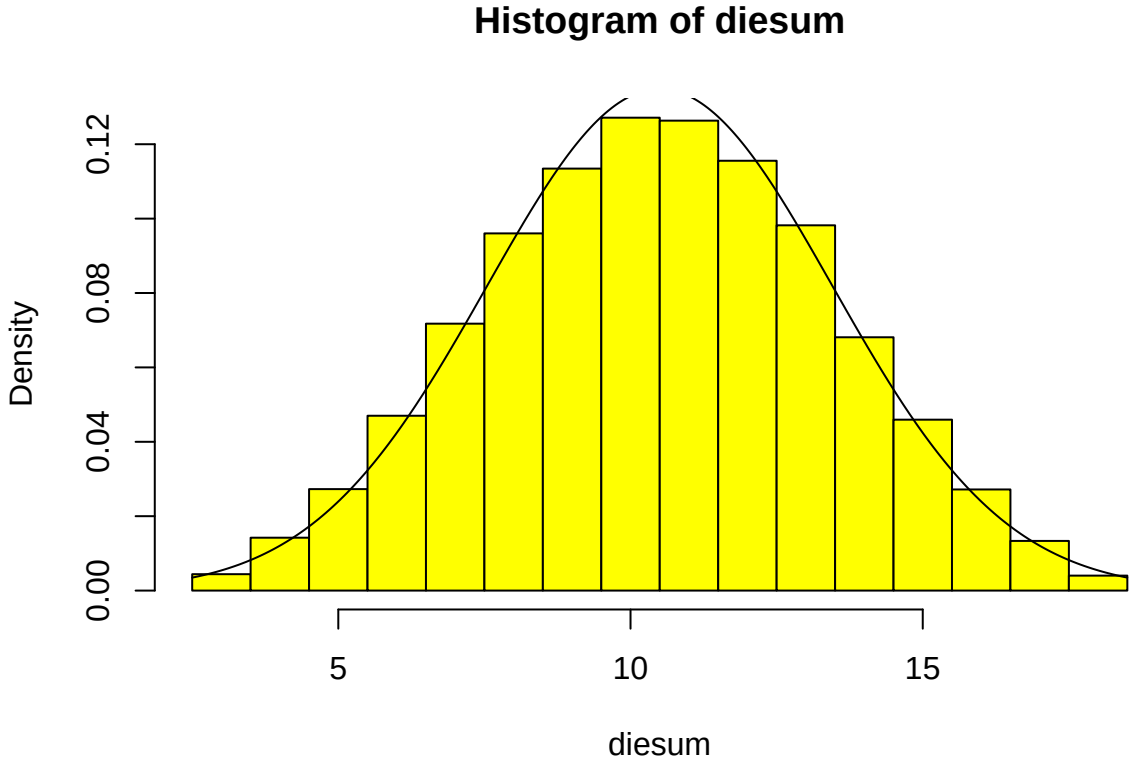
11. Modify your loop from #10 (copy and paste below) to run 100000 times (on a 100000-element vector) instead of 10 times. Plot a density histogram of the result (use `freq=FALSE` in `hist`).

```
diesum = rep(0,100000)
for(i in 1:100000) { diesum[i] = sum(sample(1:6,3,replace=TRUE) ) }
hist(diesum, col='yellow', freq=FALSE, breaks=seq(2.5,18.5,1))
```



12. Overplot a normal distribution PDF (with a mean equal to 3.5*3 and a standard deviation equal to your answer from #3 times the square root of three) on top of this histogram.

```
hist(diesum, col='yellow', freq=FALSE, breaks=seq(2.5,18.5,1))
xp = seq(2.5,18.5,0.01)
lines(xp, dnorm(xp, mean=10.5, sd=sqrt(3)*1.708))
```



Simulating the t-distribution

13. Conduct a simulation as follows.
    a. Create a vector with 10000 elements (all zeroes). Call it 'ts'.
    b. Loop over the vector 10000 times. During each iteration:
        i. Randomly draw three numbers from a standard normal distribution.
        ii. Calculate their mean m, standard deviation s, and standard error SE.
        iii. Calculate Student's t (= m/SE), and store this in the i^{th} element of ts.

(Note: You can also conduct this simulation with vector operations.)

```
ts = rep(0,10000)
for(i in 1:10000) {
  samples = rnorm(3)
  m = mean(samples)
  s = sd(samples)
  SE = s/sqrt(3)
  ts[i] = m/SE
}
```

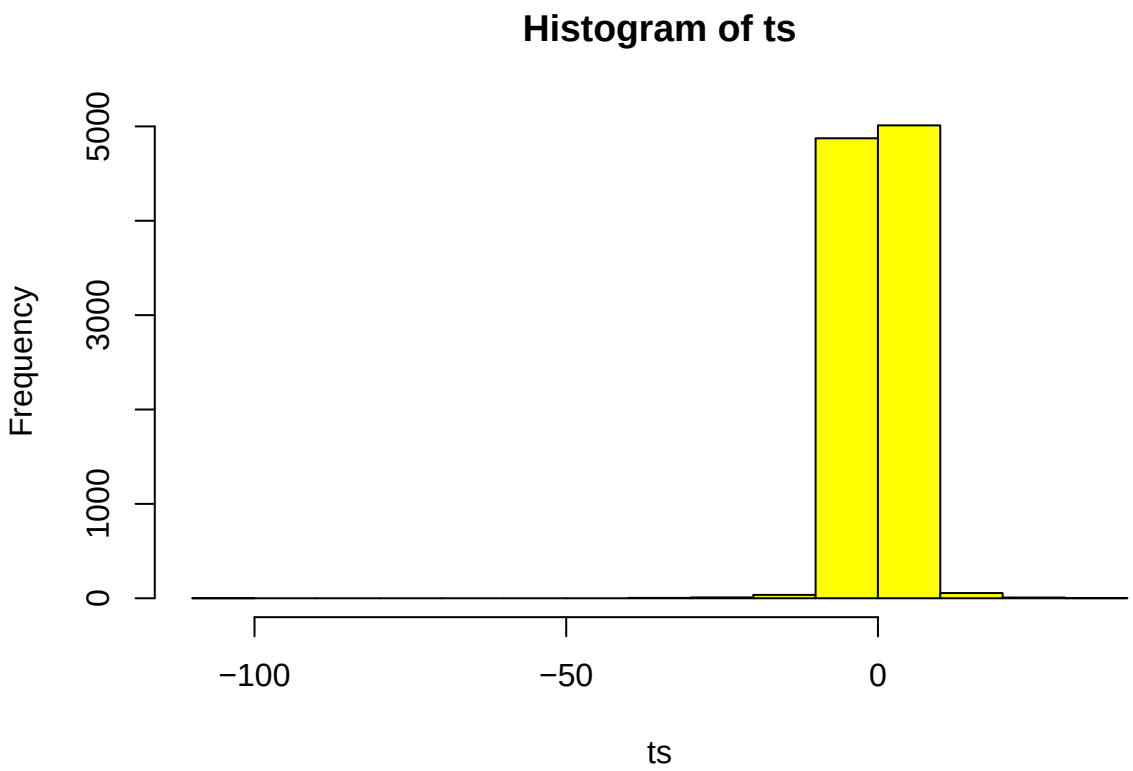
14. Print out the first 50 elements of ts.

```
ts[1:50]
```

```
##  [1]  0.91080486  1.87409464  0.03784915  0.68435523 -0.96184831  0.86101549
##  [7]  0.25815960 -1.16437467  0.30405038 -1.29819173 -0.33781594 -0.67379875
## [13]  2.86170679 -0.40398070 -0.24856401  3.74156945 -1.28401625  3.79587094
## [19]  1.26181799  1.77025177  2.23733700  0.11752149  0.23358404  1.51118621
## [25] -1.59941709  0.13133580 -0.32328700 -0.35069678 -0.29365497  0.28369147
## [31] -0.30906020 -0.03457930  1.04569660  1.81886360 -0.61539809 -0.24462140
## [37]  1.47063344  0.83113939 -1.44586127 -2.03034253 -2.53233925 -4.00459135
## [43] -0.19155526 -0.79816274 -0.79620710 -3.07522107 -0.59529859 -0.67814941
## [49] -1.19107541 -0.55734175
```

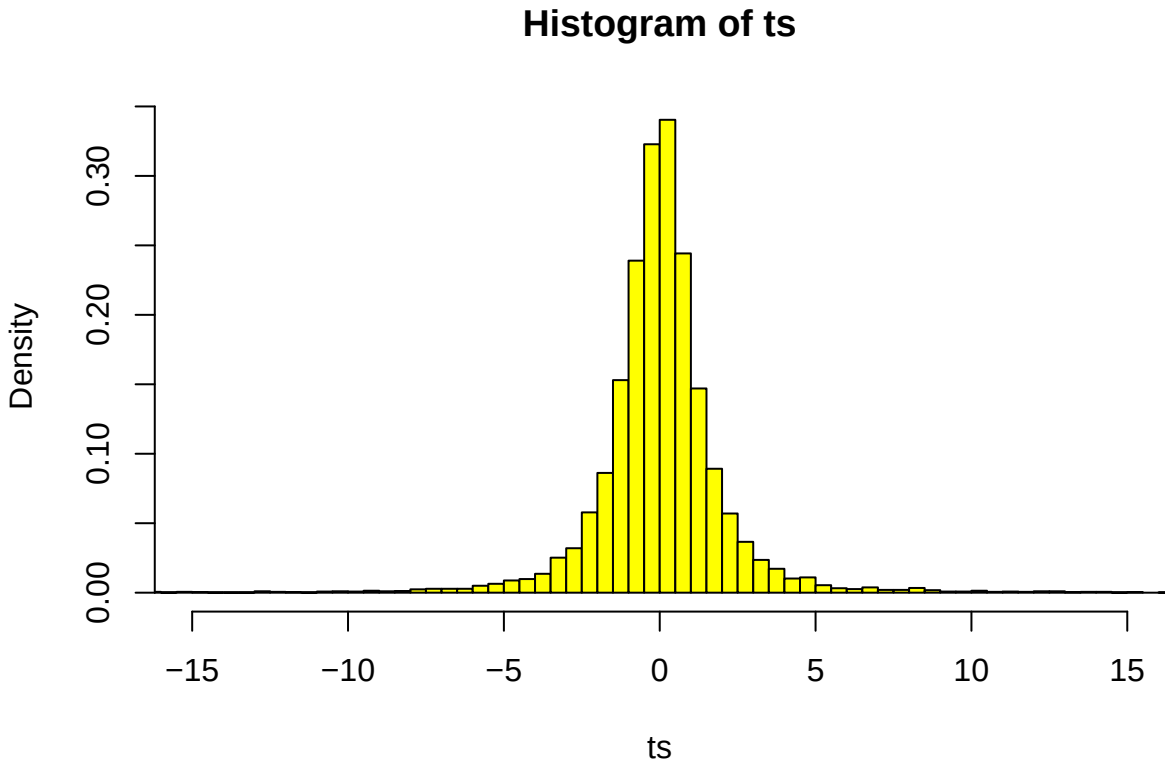
15. Plot a density histogram of ts. (You'll probably find it's not particularly informative since most values are clustered close to zero.)

```
hist(ts, col='yellow')
```



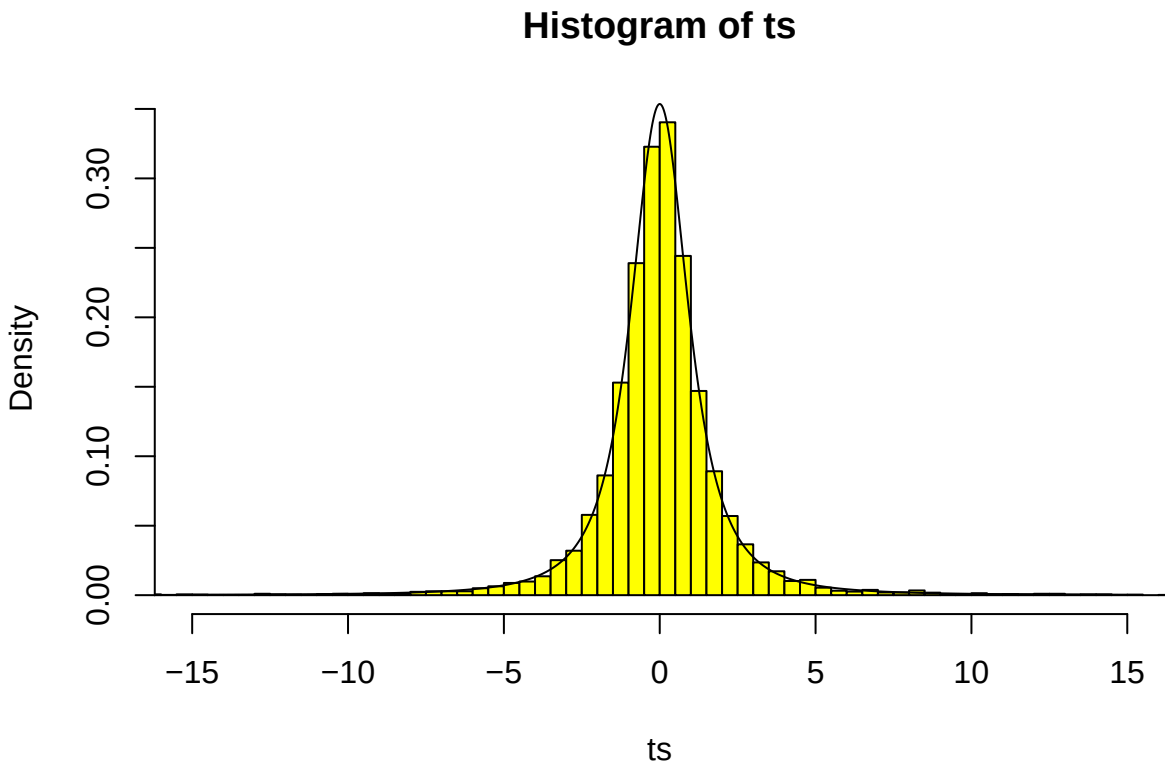
16. Plot a more informative density histogram that spans from -15 to +15 on the x-axis, with a bin spacing of 0.5. (Note: you may have to set the limits of the histogram bins to a value much wider than -15 to +15.)

```
hist(ts, col='yellow', breaks=seq(-500,500,0.5), xlim=c(-15,15), freq=FALSE)
```



17. Overplot a t distribution with 2 degrees of freedom on the histogram above. (Remember that you'll need to set up an x-axis plotting variable using seq first. If running from the Rmakdown window you'll need to repeat the histogram plotting command from your previous answer.)

```
hist(ts, col='yellow', breaks=seq(-500,500,0.5), xlim=c(-15,15), freq=FALSE)
xp = seq(-15,15,0.01)
lines(xp, dt(xp,2), type='l')
```



18. How many out of your 10000 simulation trial values of ts have an absolute value $|t| > 3$? Compare this to the predicted value based on the CDF of the t-distribution above, which is given in R by `2*(1-pt(3,2))*10000`.

```
length(which(abs(ts) > 3)) # count the values which are |t|>3
```

```
## [1] 974
```

```
sum(abs(ts) > 3)           # another way to do the above, using the fact that TRUE is stored as 1
```

```
## [1] 974
```

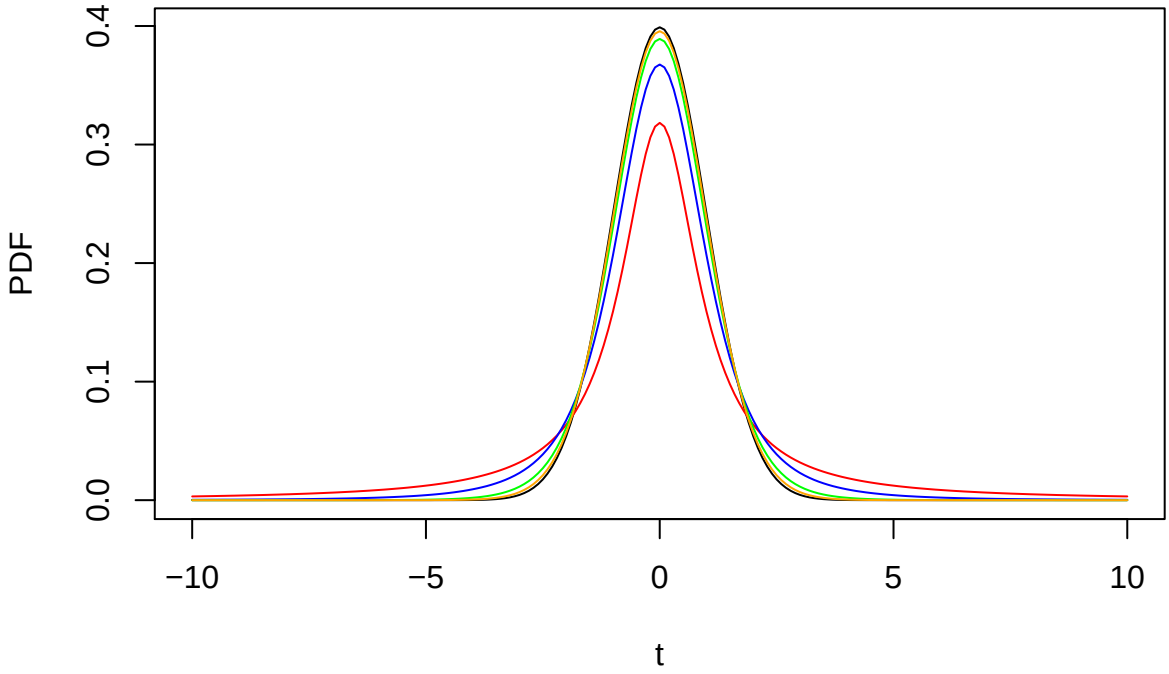
```
2*(1-pt(3,2))*10000
```

```
## [1] 954.6597
```

More on the t-distribution

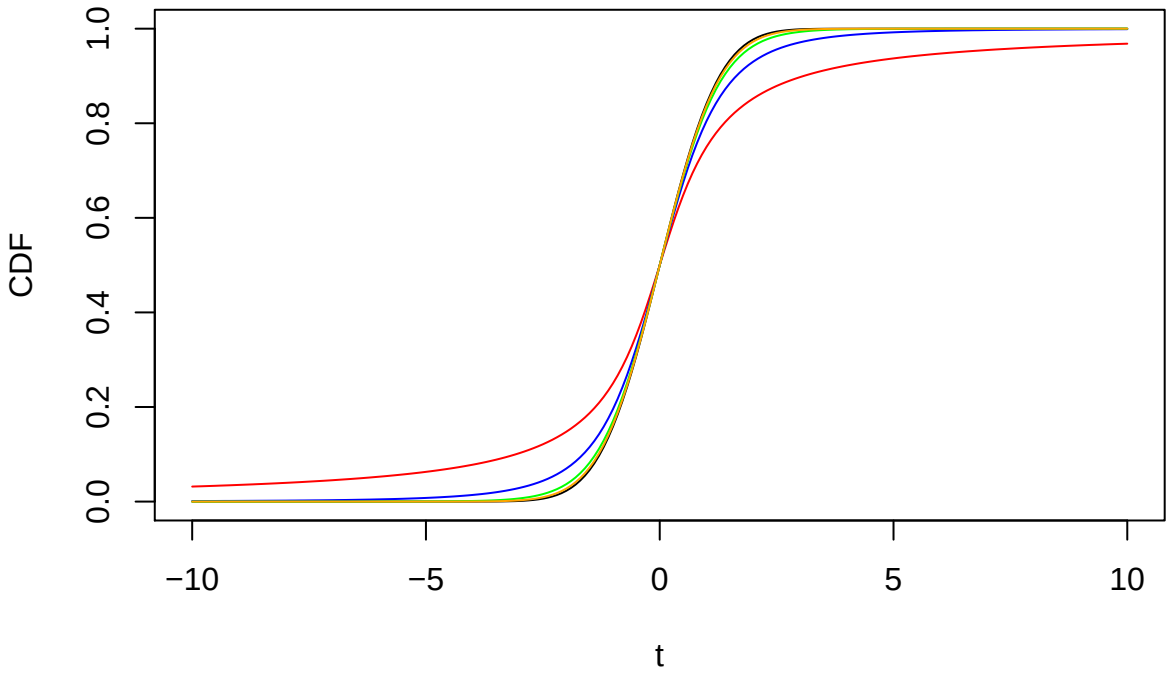
19. Produce a new plot. Plot the standard normal distribution PDF. Then overplot the t-distribution density function for 1 degree of freedom, 3 degrees of freedom, 10 degrees of freedom, and 30 degrees of freedom, each with a different colour. How do they differ? (Make sure to use an appropriate x-axis range.)

```
xp = seq(-10,10,0.1)
plot(xp, dnorm(xp), type='l', xlab='t', ylab='PDF')
lines(xp, dt(xp,1), col='red')
lines(xp, dt(xp,3), col='blue')
lines(xp, dt(xp,10), col='green')
lines(xp, dt(xp,30), col='orange')
```



20. Plot the *cumulative* distribution function for the normal distribution and for the t-distribution with the same four degree-of-freedom parameters ($\nu = 1, 3, 10, 30$) above, using the same colours.

```
xp = seq(-10,10,0.1)
plot(xp, pnorm(xp), type='l', xlab='t', ylab='CDF')
lines(xp, pt(xp,1), col='red')
lines(xp, pt(xp,3), col='blue')
lines(xp, pt(xp,10), col='green')
lines(xp, pt(xp,30), col='orange')
```



Normal inference and confidence intervals

21. Find the file with the results from the student survey used in last week's lab session (or download it again from Canvas). Read it in from disk as a dataframe (use `read.csv`) and make sure there are no NA values in the height column (use `is.na` and a logical subscript to remove any, if present). Finally, sort the (entire) dataframe according to each student's estimate of the average height of the class (use `order`).

```
sdata = read.csv('survey.csv')
sdata = sdata[!is.na(sdata$height),]
sdata = sdata[order(sdata$av_height_est),]
```

22. How many students provided a value for their height?

```
nrow(sdata)
```

```
## [1] 30
```

```
length(sdata$height)  # either works
```

```
## [1] 30
```

23. Calculate the mean height of students who took the survey and provided this value. Do you think this is precisely the same as the mean height of *all* LJMU students?

```
mean(sdata$height)
```

```
## [1] 67.7
```

It is unlikely to be precisely the same (primarily due to sampling error, though of course it is also possible that students in this class are not representative of LJMU students).

24. Suppose we hypothesize that the mean height of all LJMU students is 5'11" (= 71 inches), and we assume that the survey students can be treated as a random sampling of the broader population of

all LJMU students. Can you rule out our hypothesis based on the survey data? To do this, try the following:

a. Calculate (and print out) the value of t for this hypothesis using the formula given in lecture.
b. Determine how many degrees of freedom are present in the calculation of the sample mean (#23).
c. Determine the probability of generating a value of t "as or more extreme" than what we measured above, using the function `pt` and a two-tailed test. Is the hypothesis ruled out (at $\alpha = 0.05$)?

```
n = nrow(sdata)
t = (mean(sdata$height)-71)/(sd(sdata$height)/sqrt(n))
t
```

```
## [1] -4.649303
```

```
dof = n - 1
dof
```

```
## [1] 29
```

```
2*(1-pt(abs(t), dof))
```

```
## [1] 6.716724e-05
```

Yes, the hypothesis is ruled out (since $p < 0.05$).

25. Check your answers to #24 using the convenience function `t.test`.

```
t.test(sdata$height, mu=71)
```

```
##
##  One Sample t-test
##
## data:  sdata$height
## t = -4.6493, df = 29, p-value = 6.717e-05
## alternative hypothesis: true mean is not equal to 71
## 95 percent confidence interval:
##  66.24833 69.15167
## sample estimates:
## mean of x
##      67.7
```

26. As part of the survey, students were asked to estimate what the mean height of students would be. Treating these estimates as hypotheses about the student population more broadly, how many of these hypotheses can be ruled out (at $\alpha = 0.05$) using the survey data? How many are consistent with the data? Don't use a confidence interval (yet): re-use your code from #24, but replace the guess of 71 inches with the actual student guesses from the survey, using either a loop or a vector calculation.

```
t = (mean(sdata$height)-sdata$av_height_est)/(sd(sdata$height)/sqrt(n))
p = 2*(1-pt(abs(t), dof))
p > 0.05
```

```
##  [1] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
## [13] FALSE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE FALSE FALSE FALSE
## [25] FALSE FALSE FALSE FALSE FALSE    NA
```

```
sum(p >= 0.05 & !is.na(t)) # number allowed
```

```
## [1] 8
```

```
sum(p < 0.05  & !is.na(t))  # number ruled out
```

```
## [1] 21
```

27. Use the Student t quantile function to calculate the 95% confidence interval describing the range of (hypothetical) mean population heights consistent with the survey data. Confirm that your values agree with the values given by `t.test`.

```
mean(sdata$height) + (sd(sdata$height)/sqrt(n)) * qt(c(0.025,0.975),dof)
```

```
## [1] 66.24833 69.15167
```

```
# should agree with:
t.test(sdata$height)$conf.int
```

```
## [1] 66.24833 69.15167
## attr(,"conf.level")
## [1] 0.95
```

28. Confirm that this confidence interval includes all the "correct" student guesses, and excludes all the "incorrect" guesses. (The best way to do this is to write a loop in R that prints out each student guess, followed by the TRUE/FALSE result of a logical operation indicating whether that guess is permitted by the data.)

```
for(i in 1:n) {
    print(paste(sdata$av_height_est[i], p[i] > 0.05) )
}
```

```
## [1] "59 FALSE"
## [1] "60 FALSE"
## [1] "60 FALSE"
## [1] "60 FALSE"
## [1] "60 FALSE"
## [1] "64 FALSE"
## [1] "65 FALSE"
## [1] "65 FALSE"
## [1] "65 FALSE"
## [1] "65 FALSE"
## [1] "66 FALSE"
## [1] "66 FALSE"
## [1] "66 FALSE"
## [1] "67 TRUE"
## [1] "67 TRUE"
## [1] "67 TRUE"
## [1] "68 TRUE"
## [1] "68 TRUE"
## [1] "68 TRUE"
## [1] "68 TRUE"
## [1] "69 TRUE"
## [1] "70 FALSE"
## [1] "70 FALSE"
## [1] "70 FALSE"
## [1] "70 FALSE"
## [1] "70 FALSE"
## [1] "70 FALSE"
## [1] "70 FALSE"
## [1] "71 FALSE"
## [1] "NA NA"
```

From the above, the only guesses that are consistent with the data at $p<0.05$ (result is TRUE) are those with a value of 67 or 68. The confidence interval extends from 66.03 to 69.09, so this is consistent.)