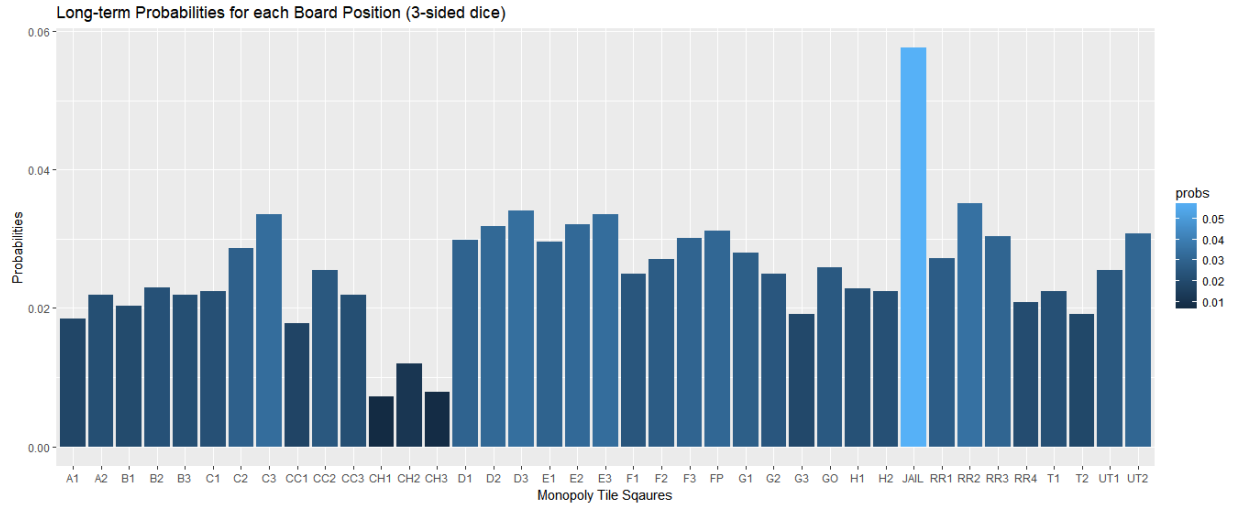
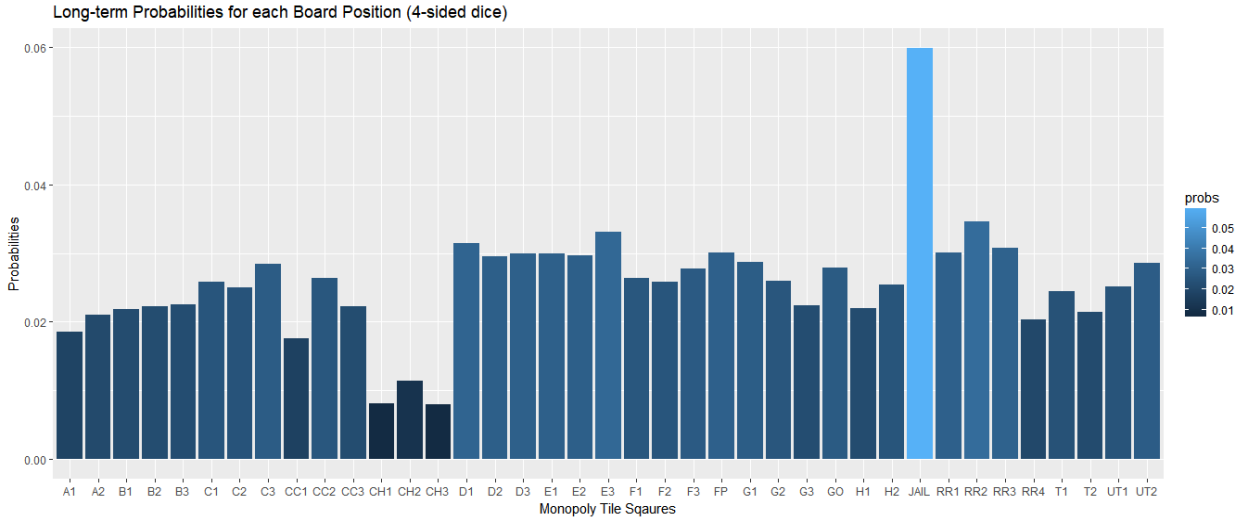
Mitchell Layton

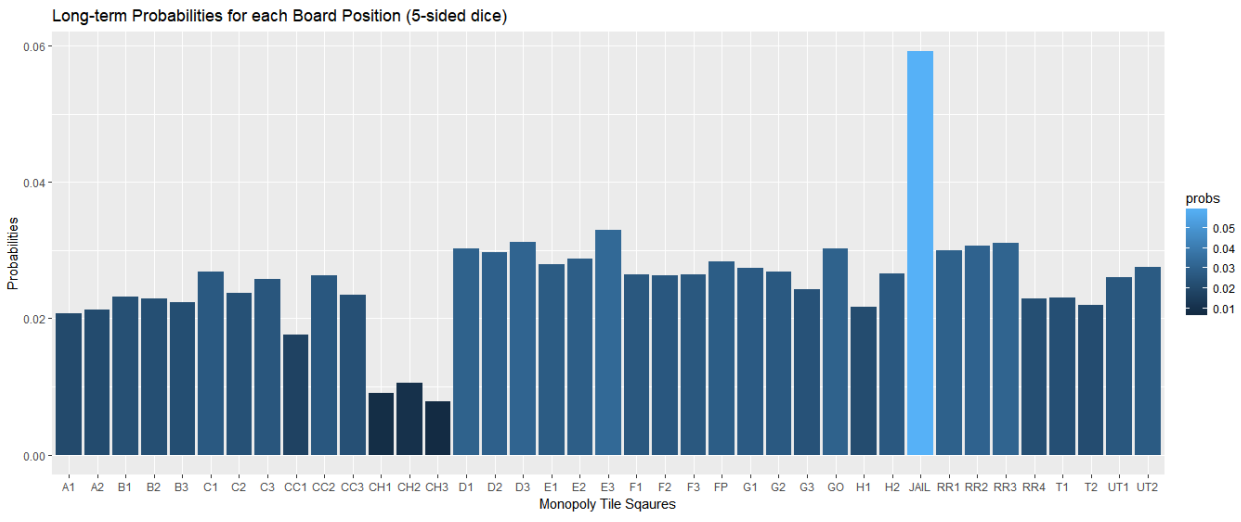
912307956

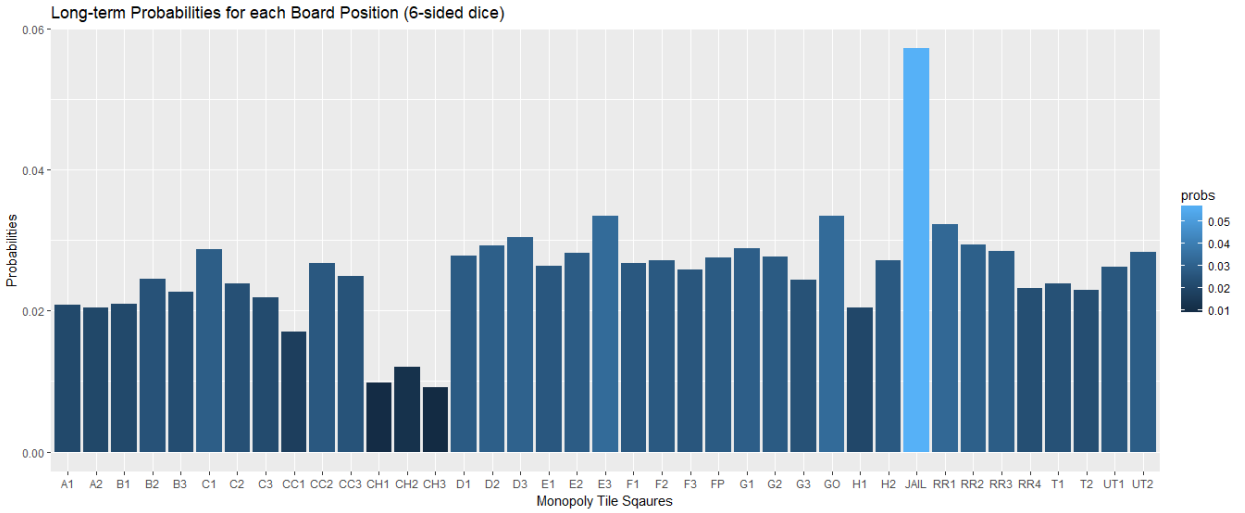
STA 141A Assignment #4 – Due Tuesday, November 21 by 5:00 pm

1) (code in appendix)









2) Above I displayed graphs for 3 up to 6-sided dice roll simulations of the probability distributions of landing on board spots. After the graphs, I will explain the most likely squares to end a turn if you play monopoly with each set of dice.

   (Note: for calculating each long term-probability for the top 3 most likely squares to end a turn on, I ran my simulation and estimate functions with each (d) parameter 3-6 and with n=500,000 to get as accurate measurement of probability as I could.)

   In my testing, when rolling the 3-sided dice through 500,000 trials, the 3 most likely squares to end a turn on are, 1. **Jail** - 5.7%; 2. **C3** -3.418%; and **RR2** – 3.414%. Next, when rolling a the 4-sided dice 500,000 trials the 3 most likely ending squares were: 1. **Jail** - 6.02%; 2. **RR2** – 3.44% 3. **E3** – 3.30%. So RR2 stays relatively close to the probability percentage seen in the 3-sided dice rolls and only increased about .03%. We notice E3 surpasses C3 by a decent jump as shown on the graph. For the 5-sided dice the top 3 are: 1. **Jail** - 5.86%; 2. **E3** - 3.29%; 3. **RR2** - 3.11%. Now we notice that E3 surpasses RR2 when going from 4 to 5-sided dice. Lastly, when rolling 6-sided dice: 1. **Jail** - 5.86%; 2. **E3** - 3.163%; 3. **GO** - 3.147%. My interpretation of adding sides to the dice in the grand scheme of long-term probabilities is that it increases the range of potential sums of both dice which in turn change the distribution and game operations. That's why with each increase in our dice sides variable (*d*), we get a new ending square, or a shift in the ending squares each time

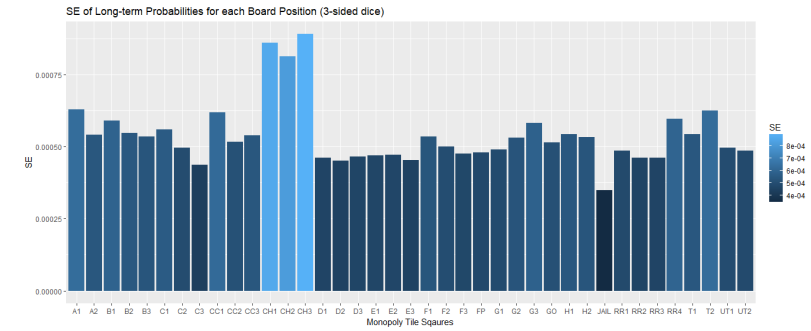
3) The standard error of our estimate for long-term probability of ending a turn in jail is **0.59**. I had calculated this by taking each of the jail observations from each replication and calculating the standard error accordingly by taking the standard deviation and dividing it by sqrt(N) where N = 1000. The interpretation of the standard error is it evaluates the precision of the means and in this case, it evaluates how precise our jail simulated observation is for the ending long-term probabilities. With a jail mean value of 588.561, and a standard error of .59, we can see that this low standard error means that we have a very precise measurement for long-term probabilities of ending a turn in jail. Furthermore, after running a t-test, I found that we are 95% confident that the true mean of the long-term probability for ending a turn in jail is between [587.32, 589.80] which is a very close interval.

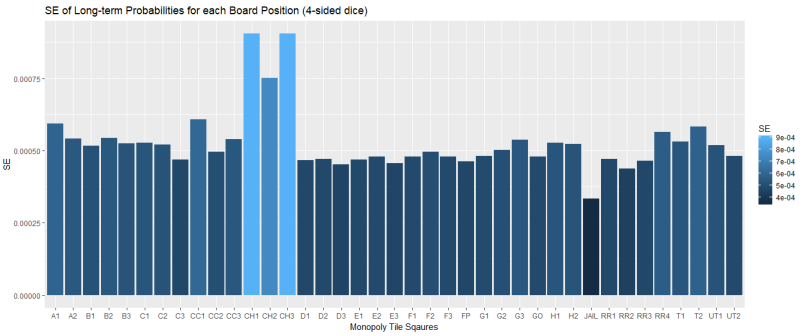
4) (a) In general, the bootstrap method is less accurate than the simulation estimates because we are only sampling with replacement from the same population of jail data. Now, when replicating that same bootstrap function 1000 times to receive 1000 different standard error estimates and take the mean of that, we are left with a closer representation and precision for the standard error of the population of simulated monopoly of n = 10,000 and d = 6. In my testing, my base R based bootstrapping function gave me a mean standard error estimate of 0.634, which is greater than our simulated estimate thus being less precise because it indicates a larger variation from our sampled population of calculated standard errors for 1000 trails.

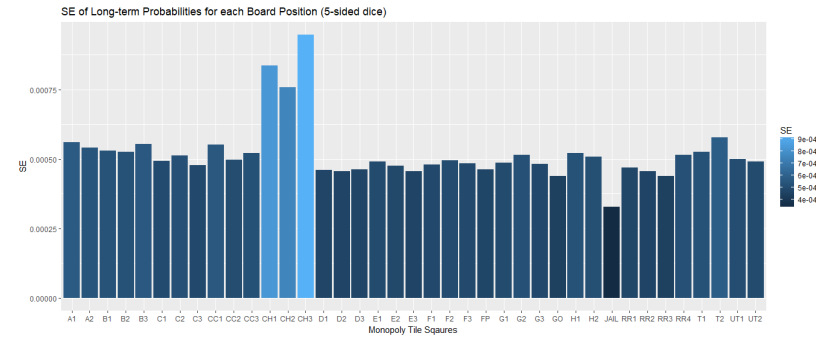
(b) The bootstrapping method is much faster to compute because in our simulation estimate, we are using are simulate_monopoly function with n=10,000 trails and replicating that process 1,000 times to extract our jail probability values for each of the 1,000 simulations of 10,000 trails. So, the CPU was put under heavy loads and took mine about 1 minute and 25 seconds to compute. Whereas the calculation of the bootstrapping method only took 1-2 seconds because it's only dealing with one population of already simulated jail values from which we sample with replacement and calculate or standard error. One can see that there is a fine line of deciding between using a simulation or a bootstrapping method all dependent upon the size of your simulations, data, time, resources, and available processing power.

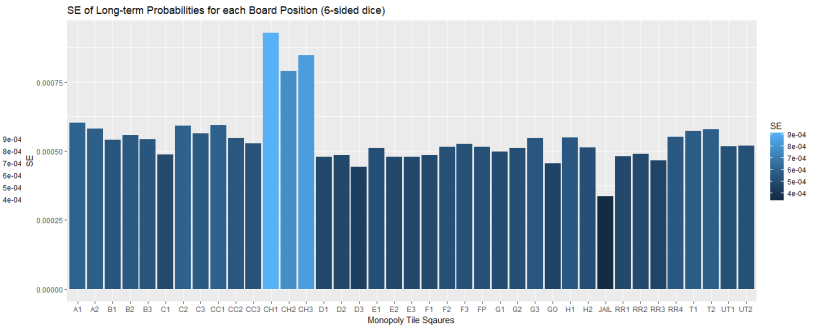
5) Above are histogram distribution breakdowns of the standard errors of the long-term probabilities for 3, 4, 5, and 6-sided dice.

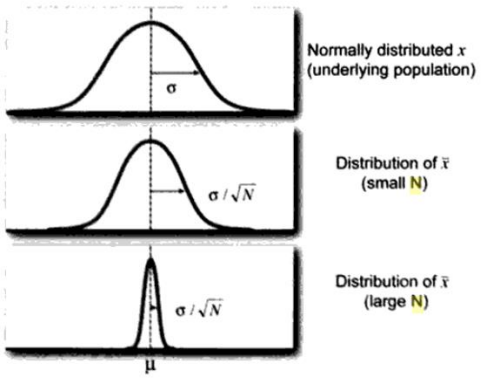
What we notice in each, is that CH1, CH2, and CH3 all have the most frequent top 3 standard errors for each dice size. When analyzing the above graphs in question 2, we may notice that the 3 lowest long-term probabilities are these 3 offenders again. So, it makes sense that the standard error of these 3 squares are the highest because the chances of being picked are much smaller than the rest of the group of long-term probabilities. Standard error here is estimating the variability between samples (in this case our generated probabilities) and the highest the standard error of our long-term probabilities, the higher the variability is among its group of probabilities for the ending squares.











6) As we have learned in previous statistical inferencing classes, the standard error calculation of large populations is (*σ/√n). So as n increases, the denominator of the calculation increases for which the equation will tend to 0 as n goes to infinity. So as n goes up, our standard error estimate goes down (inverse relationship). Thus, we are more certain in our estimate of the mean when N goes up. As shows from visual example shown from* Translational and Experimental Clinical Research (RIGHT)
7) Have not found other available group members yet, I will need to each out.

Code Appendix

```
# Preload libraries in case I use any of the following I've learned thus far
library(tidyverse)
library(ggmosaic)
library(plotly)
library(scales)
library(car)
library(RColorBrewer)
library(rio)
library(readr)
library(ggmap)
library(ggrepel)
library(data.table)
library(lubridate)
library(gridExtra)
library(plyr)
library(dplyr)
library(ggthemes)
library(grid)
library(sqldf)
library(reshape)
library(zoo)

#1) ------------------------------------------------------------------

# n = turns by a player in a game
# d = two d-sided die (from 1-6)

simulate_monopoly = function(n,d) {

   CC = seq(1,16,1)
   CC[15] = 100  # Set at index 3 for randomness and equal to 1 to rep "Advance to GO"
   CC[16] = 200
   cc = sample(sample(sample(CC))) # simulate shuffles

   CH = seq(1,16,1)
   CH[7:16] = seq(100,1000,100) # placeholders for 10/16 chances
   ch = sample(sample(sample(CH))) # simulate shuffles

   temp_funct = function() {
      nums = seq.int(1,d,1)
      sum_of_dice = sum(sample(nums, size = 2, replace = T))
      return(sum_of_dice)
   }


   card_deck_CC = function(cc) {
      pick = cc[1]
      cc = cc[2:16]
      cc[16] = pick
      x = list(pick,cc)
      return(x)
   }

   card_deck_CH = function(ch) {
      pic = ch[1]
```

```
   ch = ch[2:16]
   ch[16] = pic
   y = list(pic,ch)
   return(y)
}


dice_rolls = as.vector(replicate(n, temp_funct()))
output = vector("double", length(dice_rolls))
MAX_BOARD_POSITION = 39
current_position = 0

dice_seq = seq(1,(length(dice_rolls)+1),by=1)

for (i in dice_seq) {

      if (i == 1) {
         output[1] = 0
         output[2] = dice_rolls[1]
         output[3] = dice_rolls[1] + dice_rolls[2]

      }
      if (i >= 4) {
         output[i] = output[i-1] + dice_rolls[i-1]
         if ((output[i]) < 39) {
            output[i] = (dice_rolls[i-1] + output[i-1])
            dice_rolls[i-1] = output[i]
         }
         if (output[i] == 39) {
            output[i+1] = output[i]
         }

         if (output[i] > 39) {
            output[i] = ((output[i] %% MAX_BOARD_POSITION) - 1)
            dice_rolls[i-1] = output[i]
         }
         if (output[i] == 40) {
            output[i] = 0
         }
         if (output[i] == 30) {
            output[i] = 10
         }
         if (output[i] %in% c(2,17,33)) {
            pick = card_deck_CC(cc)
            pi = pick[[1]]
            cc = pick[[2]]
            if (pi < 20) {
               output[i] = output[i]
            }
            else if (pi == 100) {
               output[i] = 0 # Advance to GO
            }
            else
               output[i] = 10 # GO to JAIL
         }
         if (output[i] %in% c(7,22,36)) {
            pic = card_deck_CH(ch)
            p = pic[[1]]
```

```
ch = pic[[2]]
# For Go to next RR if else and UT
RR1 = 5
RR2 = 15
RR3 = 25
RR4 = 35
UT1 = 12
UT2 = 28
if (p < 10) {
   output[i] = output[i]
}
else if (p == 100) { # Advance to GO
   output[i] = 0
}
else if (p == 200) { # Go to JAIL
   output[i] = 10
}
else if (p == 300) { # Go to C1
   output[i] = 11
}
else if (p == 400) { # Go to E3
   output[i] = 24
}
else if (p == 500) { # Go to H2
   output[i] = 39
}
else if (p == 600) {# Go to RR1
   output[i] = 5
}
else if (p == 700) { # Go to next RR (railroad)
   output[i] = output[i]
   if ((output[i] >= 0) & (output[i]<= 4)) {
      output[i] = RR1
   }
   if ((output[i] >= 6) & (output[i]<= 14)) {
      output[i] = RR2
   }
   if ((output[i] >= 16) & (output[i]<= 24)) {
      output[i] = RR3
   }
   if ((output[i] >= 26) & (output[i]<= 34)) {
      output[i] = RR4
   }
   if ((output[i] >= 36) & (output[i]<= 39)) {
      output[i] = RR1
   }

}
else if (p == 800) { # Go to next RR (railroad)
   output[i] = output[i]
   if ((output[i] >= 0) & (output[i]<= 4)) {
      output[i] = RR1
   }
   if ((output[i] >= 6) & (output[i]<= 14)) {
      output[i] = RR2
   }
   if ((output[i] >= 16) & (output[i]<= 24)) {
      output[i] = RR3
```

```
                }
                if ((output[i] >= 26) & (output[i]<= 34)) {
                  output[i] = RR4
                }
                if ((output[i] >= 36) & (output[i]<= 39)) {
                  output[i] = RR1
                }
              }
              else if (p == 900) { # Go to next UT (utility)
                output[i] = output[i]
                if ((output[i] >= 0) & (output[i]<= 11)) {
                  output[i] = UT1
                }
                if ((output[i] >= 13) & (output[i]<= 27)) {
                  output[i] = UT2
                }
                if ((output[i] >= 29) & (output[i]<= 39)) {
                  output[i] = UT1
                }
              }
              else
                output[i] = output[i] - 3


            }


          }

    } # for i

    if (length(output) > (n+1)) {
      output[1:(length(output)-1)]
    }
    else
      return(output)
}

output = simulate_monopoly(10000,6)
output # put in arguments (n,d) for function here
# a length n + 1 vector of positions, encoded as numbers from 0 to 39.


#2) ----------------------------------------------------------------


estimate_monopoly = function(x) {

    values = as.data.table(x)
    names(values) = "Values"

    positions = as.data.table(c(
      "GO", "A1", "CC1", "A2", "T1", "RR1", "B1", "CH1", "B2", "B3", "JAIL",
      "C1", "UT1", "C2", "C3", "RR2", "D1", "CC2", "D2", "D3", "FP",
      "E1", "CH2", "E2", "E3", "RR3", "F1", "F2", "UT2", "F3", "G2J",
      "G1", "G2", "CC3", "G3", "RR4", "CH3", "H1", "T2", "H2"
    ))
```

```
  spots = as.data.table(c(seq(0,39,1)))
  df = (cbind(positions,spots))
  names(df) = c("Square_Names","Values")

  merged_data = merge(values, df, by = "Values")
  table_merged = as.data.table(table(merged_data$Square_Names))
  names(table_merged) = c("squares","nums")

  # Get probabilities
  table_merged = table_merged %>%
    mutate(probs = (table_merged$nums)/sum(table_merged$nums))


    monopoly_freq = ggplot(data = table_merged, aes(x=squares,y=probs, fill = probs)) +
      labs(x = "Monopoly Tile Sqaures", title = "Long-term Probabilities for each Board Position (6-sided dice)") +
       geom_bar(stat="identity") +
       scale_y_continuous("Probabilities")
     print(monopoly_freq)
     return(table_merged)

}
  # Probabilities all based on d parameter in above function. Chance (d) sided die for below input to funciton
  tab = estimate_monopoly(output)
  G = sqldf("SELECT * FROM tab ORDER BY probs DESC")
  TOP_3 = head(G,3)
  TOP_3


#3) ----------------------------------------------------------------
  # partially referenced method from Anonymous Piazza user
  replicate_function = function(k,n) {

    table = as.data.table(numeric(k))
      for (i in 1:k){
        nums = as.data.table(table(simulate_monopoly(n,6)))
        t1 = nums$N[nums$V1 == "10"]
        table[i,] = t1
      }
    table
  }

  # Takes about 01:24.00 s to compute
  jail_values = replicate_function(1000,10000)
  names(jail_values) = "Jail"

  #computation of the standard error of the mean of JAIL occurances and amounts
  SE = sd(jail_values$Jail)/sqrt(length(jail_values$Jail))
  SE
  t.test(jail_values$Jail)



#4) ----------------------------------------------------------------

  se_boot = function(x = jail_values$Jail) {
    x.boot = sample(x, size = length(x), replace=T)
    y = sd(x.boot)/sqrt(length(x.boot)) # SE
    y
```

```
  }
  se_boot()
  se_boot.replicate = as.data.table(replicate(1000, se_boot()))
  se_boot.replicate
  t.test(se_boot.replicate)




#5) ------------------------------------------------------------------


  SE_long_term = function(x) {

    values = as.data.table(x)
    names(values) = "Values"

    positions = as.data.table(c(
      "GO", "A1", "CC1", "A2", "T1", "RR1", "B1", "CH1", "B2", "B3", "JAIL",
      "C1", "UT1", "C2", "C3", "RR2", "D1", "CC2", "D2", "D3", "FP",
      "E1", "CH2", "E2", "E3", "RR3", "F1", "F2", "UT2", "F3", "G2J",
      "G1", "G2", "CC3", "G3", "RR4", "CH3", "H1", "T2", "H2"
    ))

    spots = as.data.table(c(seq(0,39,1)))
    df = (cbind(positions,spots))
    names(df) = c("Square_Names","Values")

    merged_data = merge(values, df, by = "Values")
    table_merged = as.data.table(table(merged_data$Square_Names))
    names(table_merged) = c("squares","nums")

    # Get probs
    table_merged = table_merged %>%
      mutate(probs = (table_merged$nums)/sum(table_merged$nums))
    # Get SE
    table_merged = table_merged %>%
      mutate(SE = sd(table_merged$probs)/sqrt(table_merged$nums))

    SE_freq = ggplot(data = table_merged, aes(x=squares,y = SE, fill = SE)) +
      labs(x = "Monopoly Tile Sqaures", title = "SE of Long-term Probabilities for each Board Position (3-sided
dice)") +
        geom_bar(stat="identity") +
        scale_y_continuous("SE")
    print(SE_freq)
    return(table_merged)

  }
  # Probabilities all based on d parameter in above function. Chance (d) sided die for below input to funciton
  tab = SE_long_term(output)
  tab
```