STT 301: Apply family of functions

Shawn Santo

October 15, 2018

Introduction

Learning objectives:

- reading data into R
    - `read.csv`
- `apply` function
    - matrices
    - data frames
    - arrays
- apply family

```
library(ggplot2)
```

The apply function

Part a

Download the csv file `nevada_casino_sqft.csv` from D2L. The data set contains Nevada casino square footage by activity. Before you read the csv file into R, open it and look at how `NA` values are represented as well as if variable headers are present. Then, use `read.csv` to read the file in to R (make use of the `na.strings` argument). Save the data frame as `casino.na`. Examine the structure of the data frame `casino.na`.

```
'data.frame':   263 obs. of  10 variables:
 $ COUNTY  : int  14 4 16 2 2 2 2 2 2 16 ...
 $ AREA    : int  0 0 2 0 3 4 2 5 0 1 ...
 $ NAME    : chr  "ALAMO CASINO - MILL CITY                 " "ALAMO CASINO AT WELLS PETR
O              " "ALAMO TRAVEL CENTER                     " "ALBERTSON'S #6046
            " ...
 $ PITGAMES: int  0 0 900 0 5060 8215 48147 2125 1903 12500 ...
 $ SLOTS   : int  3500 2250 6100 400 98007 42075 86028 35950 39092 45775 ...
 $ KENO    : int  0 0 0 0 0 1680 0 0 560 1000 ...
 $ BINGO   : int  0 NA 0 0 5624 0 0 7546 9196 0 ...
 $ SPORTS  : int  0 0 0 0 14200 5100 10156 1920 4096 3648 ...
 $ POKER   : int  0 0 150 0 2109 0 5669 0 380 1891 ...
 $ TOTAL   : int  3500 NA 7150 400 125000 57070 150000 47541 55227 64814 ...
```

Part b

Use the `apply` function to

1. check how many `NA` values are in each row;
2. find the mean square footage for “PITGAMES”, “SLOTS”, “KENO”, “BINGO”, “SPORTS”, “POKER”, and “TOTAL”;
3. find the median square footage for “PITGAMES”, “SLOTS”, “KENO”, “BINGO”, “SPORTS”, “POKER”, and “TOTAL”;
4. compute the five number summary (use the function `summary`) for “PITGAMES”, “SLOTS”, “KENO”, “BINGO”, “SPORTS”, “POKER”, and “TOTAL”;

```
  [1] 0 2 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 3 0 0 0 0 0 0 2 0 0 0 0 0 0 0 0 0
 [36] 0 0 0 0 0 0 0 0 0 2 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 2
 [71] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 2 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
[106] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
[141] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
[176] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
[211] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
[246] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
```

```
  PITGAMES      SLOTS       KENO      BINGO     SPORTS      POKER 
 4084.6654 24738.3194   223.6489   981.5753  1590.7395   663.9848 
     TOTAL 
32904.1634 
```

```
PITGAMES    SLOTS     KENO    BINGO   SPORTS    POKER    TOTAL 
     500    11963        0        0        0        0    13000 
```

```
$PITGAMES
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max. 
      0       0     500    4085    4456   65634 

$SLOTS
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max. 
      0    4530   11963   24738   37995  130000 

$KENO
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.    NA's 
    0.0     0.0     0.0   223.6     0.0  6800.0       1 

$BINGO
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.    NA's 
    0.0     0.0     0.0   981.6     0.0 18278.0       4 

$SPORTS
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.    NA's 
      0       0       0    1591    1480   21411       2 

$POKER
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max. 
      0       0       0     664     600   15170 

$TOTAL
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.    NA's 
      2    5400   13000   32904   47541  186187       6 
```

Part c

Download the same data set (this does not have `NA` values) from "http://www.stat.ufl.edu/~winner/data/nevada_casino_sqft.csv (http://www.stat.ufl.edu/~winner/data/nevada_casino_sqft.csv)". Use this URL directly in the `read.csv` function. Set `stringsAsFactors = FALSE`. Save the data frame as `casino`. Examine the structure of `casino`.

```
'data.frame':   263 obs. of  10 variables:
 $ COUNTY  : int  14 4 16 2 2 2 2 2 2 16 ...
 $ AREA    : int  0 0 2 0 3 4 2 5 0 1 ...
 $ NAME    : chr  "ALAMO CASINO - MILL CITY               " "ALAMO CASINO AT WELLS PETR
O             " "ALAMO TRAVEL CENTER                     " "ALBERTSON'S #6046
            " ...
 $ PITGAMES: int  0 0 900 0 5060 8215 48147 2125 1903 12500 ...
 $ SLOTS   : int  3500 2250 6100 400 98007 42075 86028 35950 39092 45775 ...
 $ KENO    : int  0 0 0 0 0 1680 0 0 560 1000 ...
 $ BINGO   : int  0 0 0 0 5624 0 0 7546 9196 0 ...
 $ SPORTS  : int  0 0 0 0 14200 5100 10156 1920 4096 3648 ...
 $ POKER   : int  0 0 150 0 2109 0 5669 0 380 1891 ...
 $ TOTAL   : int  3500 2250 7150 400 125000 57070 150000 47541 55227 64814 ...
```

Part d

Use the `with` function to

1. create a table for the variable `AREA`;

2. compute the mean `SPORTS` square footage for when the `SPORTS` square footage is greater than 0;
3. compute the median `POKER` square footage for when both the `POKER` and `SPORTS` square footage are greater than 0.

```
AREA
  0   1   2   3   4   5   6
112  42  50  16   9  31   3
```

```
[1] 4613.144
```

```
[1] 1965
```

Part e

Write a function that computes something simple. Use `apply` to apply your written function to columns 4 through 10 of the `casino` data frame.

Central Limit Theorem and the apply function

Here you will write a function and then use the `apply` function so you can demonstrate the Central Limit Theorem through a simulation. Assume our population distribution is the Poisson distribution. The goal is for your function to create similar plots as you see below. Thus, the sampling distribution of the sample mean should become more mound-shaped and symmetric as the sample size increases.

Some helpful hints for writing your function:

- have your function take two arguments: sample size - `n` and Poisson parameter - `lambda`
- use `rpois` to generate random variables from the Poisson distribution
- use 10000 replications for the user specified sample size (thus the function will need to generate n * 10000 random variables)
- store the generated Poisson random variables in an n x 10000 matrix (use the R function `matrix` and don't forget to specify the matrix dimensions)
- use `apply` to compute the mean of each column of the matrix
- plot the distribution of the sample means using the `geom_bar` geom function from `ggplot2`
- use `set.seed(1360)` inside your function, before you generate the random variables, to have your plots match exactly to the below plots

```
pois.clt(n = 1, lambda = 1)
```

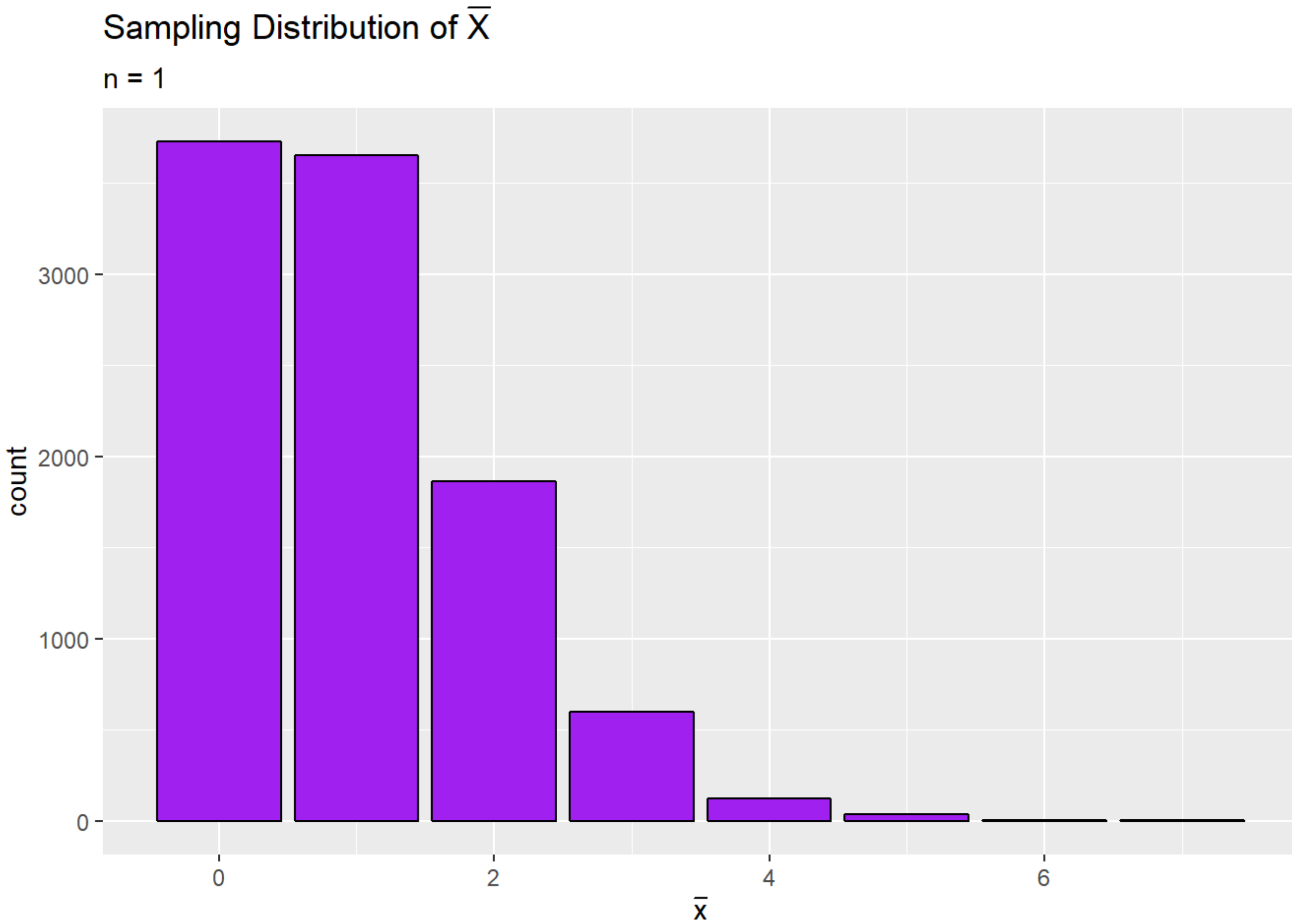
```
pois.clt(n = 2, lambda = 1)
```

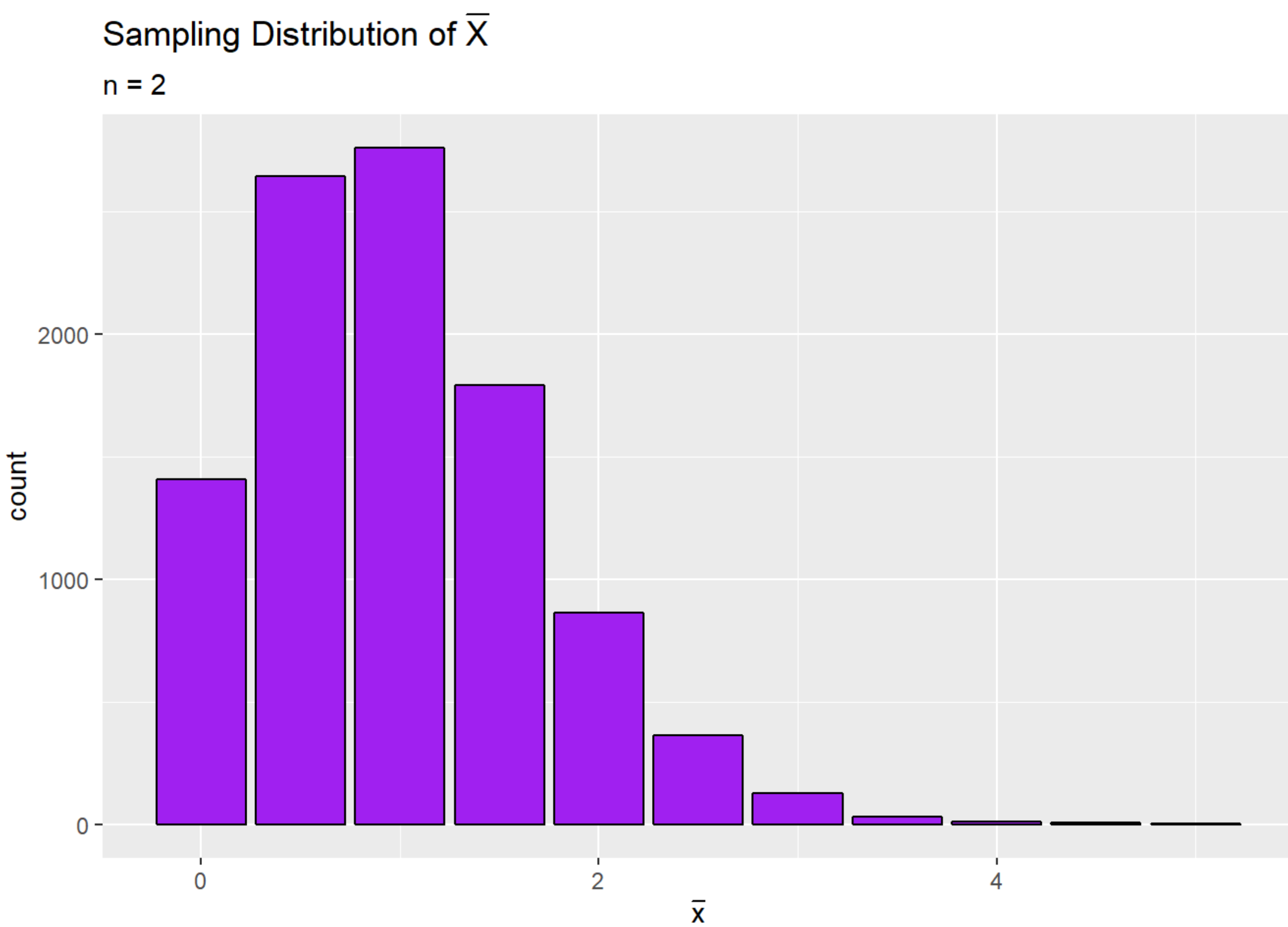
```
pois.clt(n = 5, lambda = 1)
```

```
pois.clt(n = 10, lambda = 1)
```

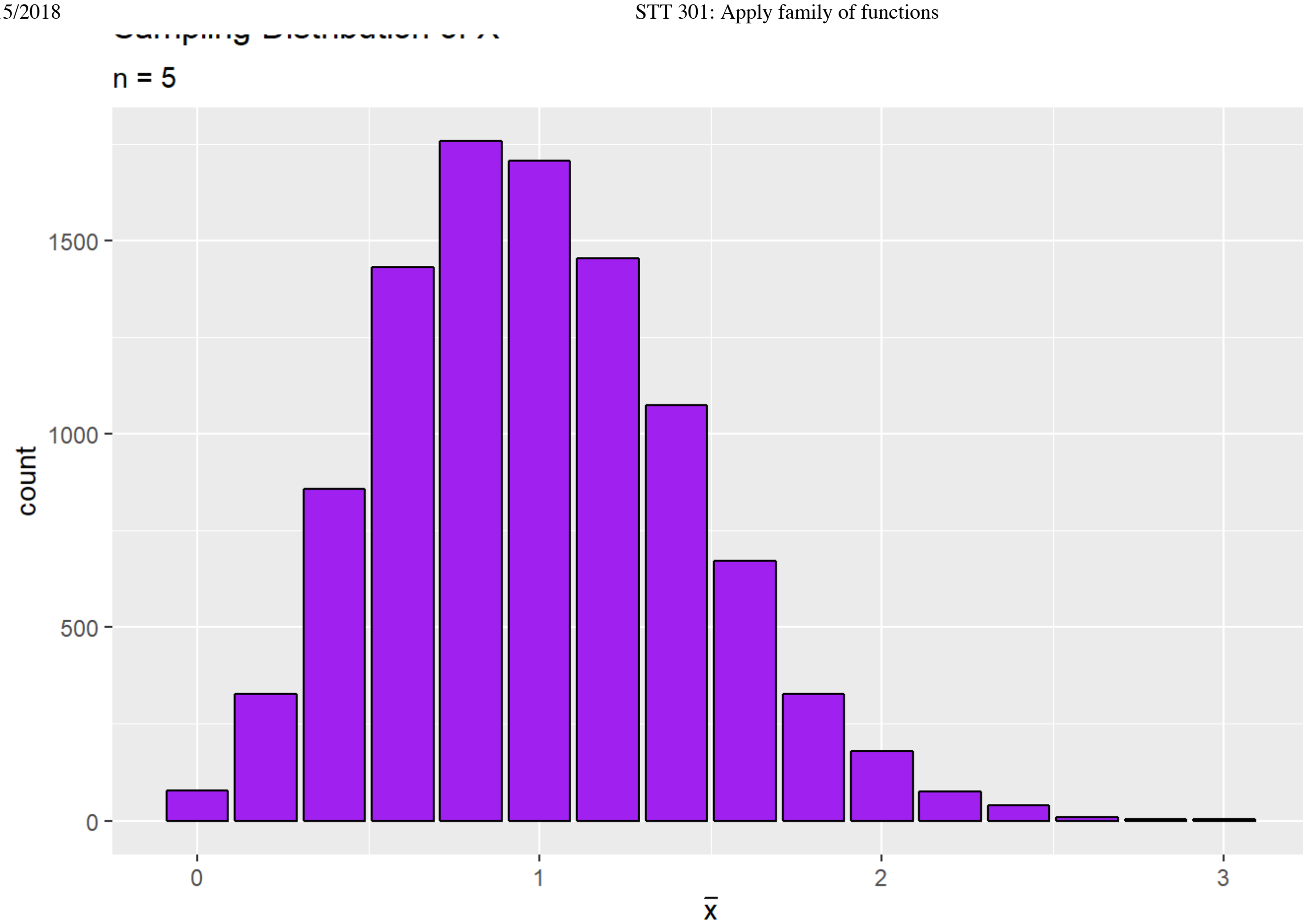
```
pois.clt(n = 30, lambda = 1)
```

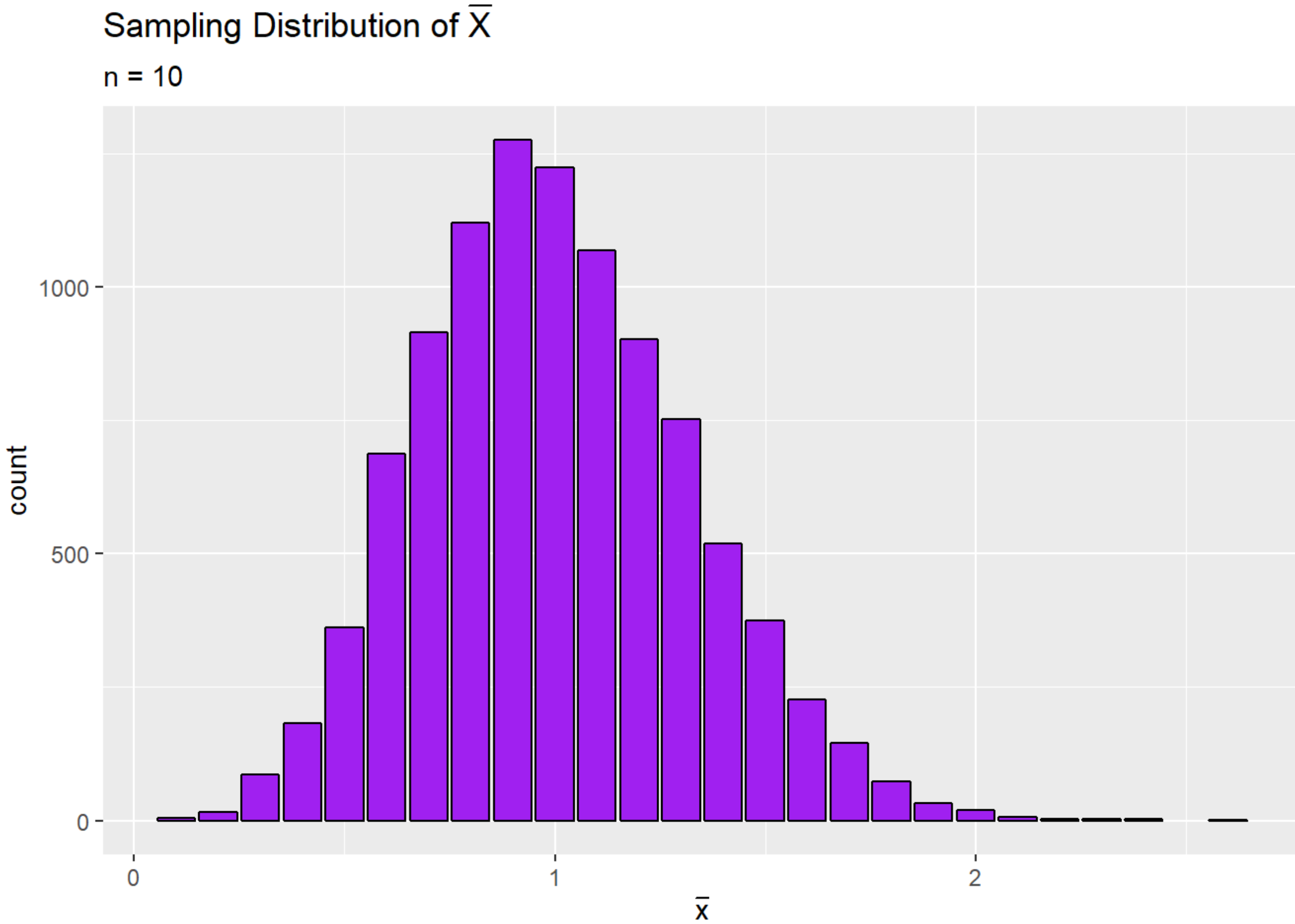
```
pois.clt(n = 50, lambda = 1)
```

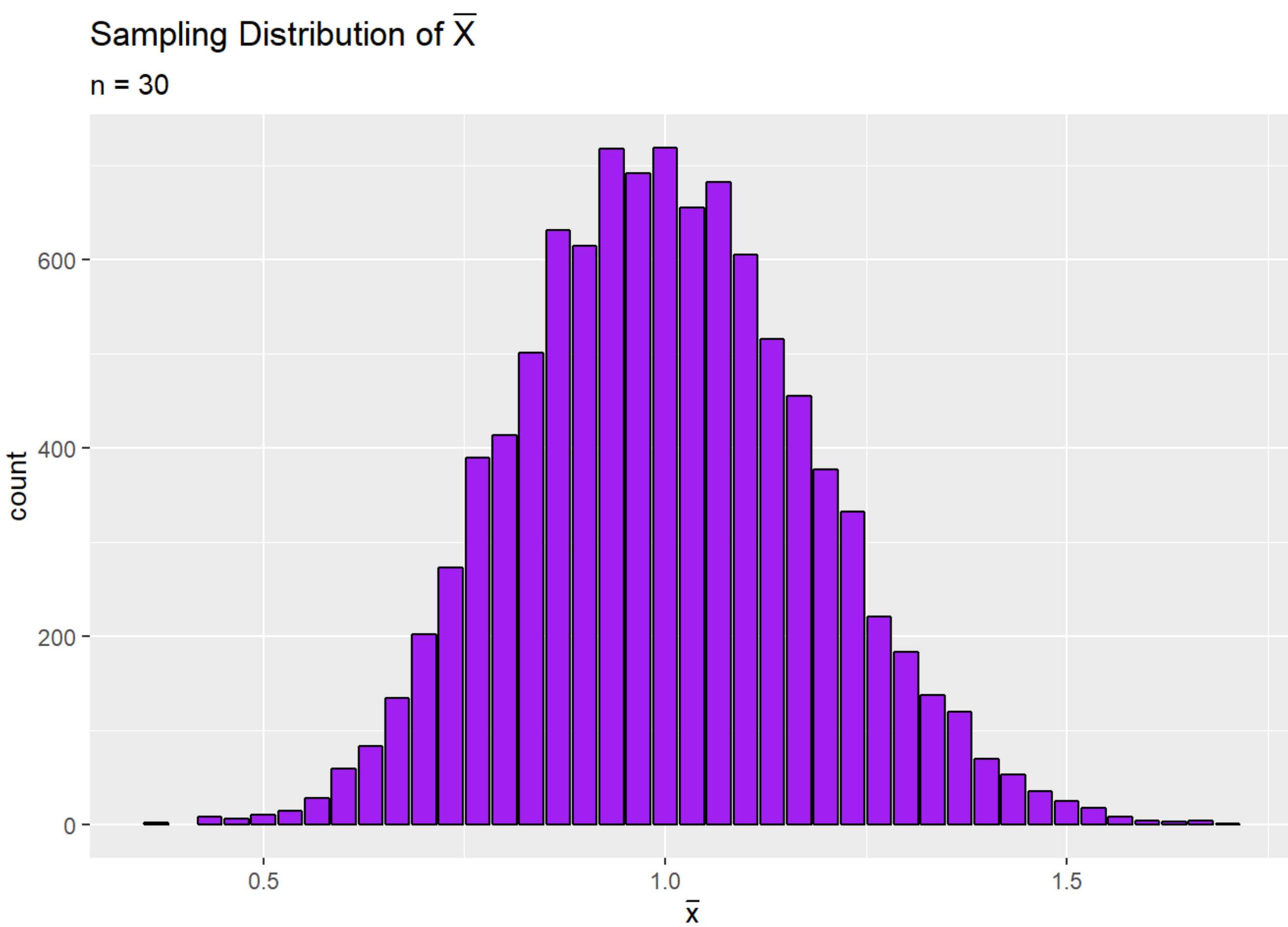


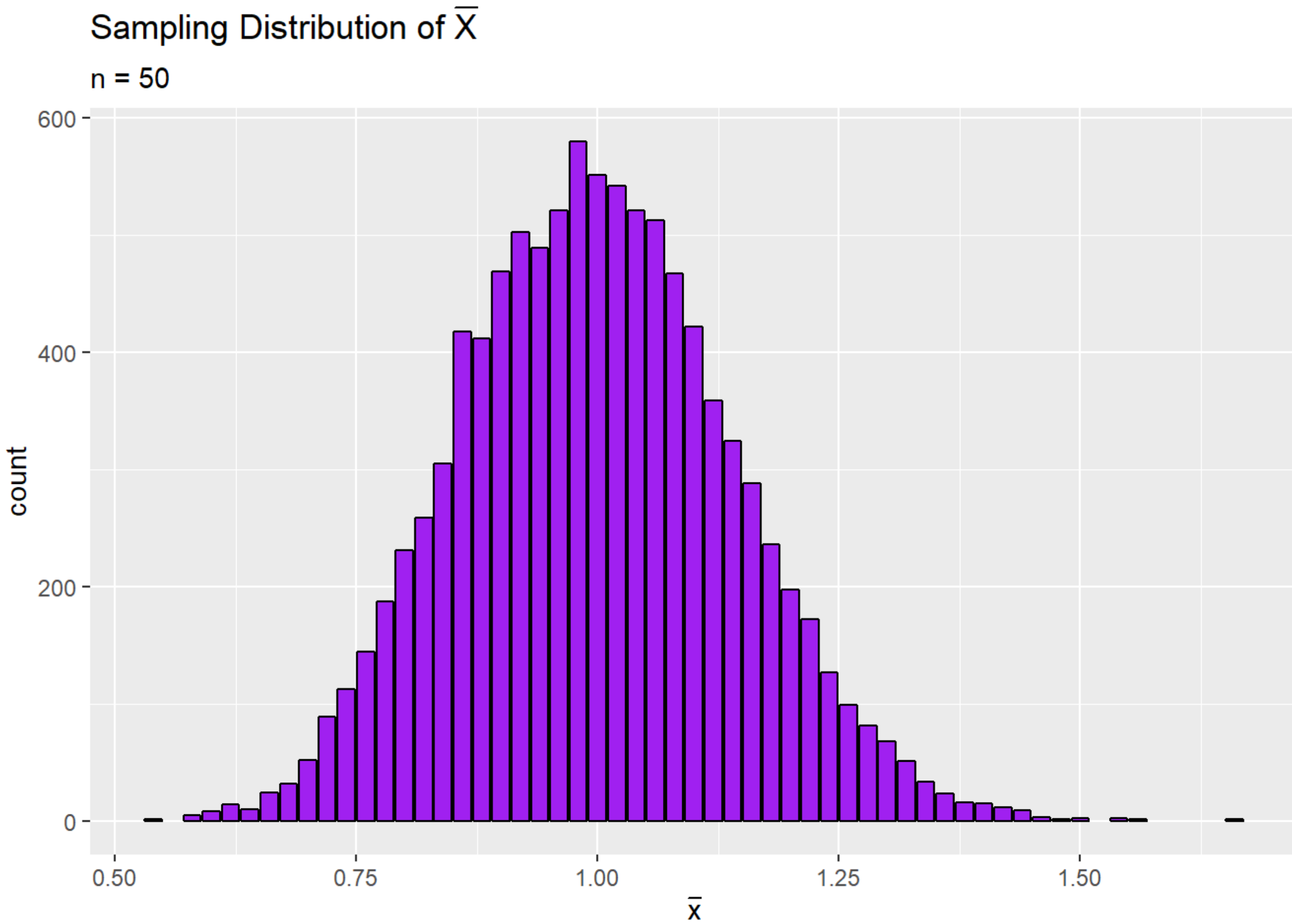


Sampling Distribution of $\overline{X}$


n = 5
count
1500
1000
500
0
0
1
2
3
$\bar{x}$


Sampling Distribution of $\overline{X}$
n = 10
count
1000
500
0
0
1
2
$\bar{x}$


Sampling Distribution of $\overline{X}$
n = 30
count
600
400
200
0
0.5
1.0
1.5
$\bar{x}$



Apply with arrays

Understand each of the below examples using `apply` on a three dimensional array. Run the code line-by-line to see what happens.

```
# create a 2 x 2 x 3 array that contains the numbers 1 - 12

my.array <- array(data = c(1:12), dim = c(2,2,3))

# view the array

my.array

# apply sum over 1 dimension

apply(my.array, 1, sum)
apply(my.array, 2, sum)
apply(my.array, 3, sum)

# apply sum over multiple dimensions

apply(my.array, c(1,2), sum)
apply(my.array, c(1,3), sum)
apply(my.array, c(2,3), sum)
apply(my.array, c(3,1), sum)
apply(my.array, c(3,2), sum)
```

A summary of the `a,l,s,t apply` functions

Command	Description
`apply(X, MARGIN, FUN, ...)`	Obtain a vector/array/list by applying `FUN` along the specified `MARGIN` of an array or matrix `X`
`lapply(X, FUN, ...)`	Obtain a list by applying `FUN` to the elements of a list `X`
`sapply(X, FUN, ...)`	Simplified version of `lapply`. Returns a vector/array instead of list.
`tapply(X, INDEX, FUN, ...)`	Obtain a table by applying `FUN` to each combination of the factors given in `INDEX`

- these functions are good alternatives to loops
- they are typically more efficient than loops (often run considerably faster on large data sets)
- take practice to get used to, but make analysis easier to debug and less prone to error when used effectively
- you can always type `example(function)` to get code examples (E.g., `example(apply)`)