R objects, Workflow, and Functions

Vectors

Create a vector!

```
set.seed(42)
my_unif <- runif(30)
is.vector(my_unif)
```

```
[1] TRUE
```

Sebset that object

```
my_unif[1:10]
```

```
 [1] 0.9148060 0.9370754 0.2861395 0.8304476 0.6417455 0.5190959 0.7365883
 [8] 0.1346666 0.6569923 0.7050648
```

```
my_unif[c(1:3, 15:17)]
```

```
[1] 0.9148060 0.9370754 0.2861395 0.4622928 0.9400145 0.9782264
```

Sort the vector

```
sort(my_unif)
```

```
 [1] 0.08243756 0.11748736 0.13466660 0.13871017 0.25542882 0.28613953
 [7] 0.39020347 0.44696963 0.45774178 0.46229282 0.47499708 0.51421178
[13] 0.51909595 0.56033275 0.64174552 0.65699229 0.70506478 0.71911225
[19] 0.73658831 0.83044763 0.83600426 0.90403139 0.90573813 0.91480604
[25] 0.93467225 0.93707541 0.94001452 0.94666823 0.97822643 0.98889173
```

Create a vector with strings in it.(number first, lower case then, last upper case)

```
char_vec <- c('daf', "adf", "E2", '13da')
sort(char_vec)
```

```
[1] "13da" "adf"  "daf"  "E2"
```

Data Frame

```
data(trees)
trees
```

```
   Girth Height Volume
1    8.3     70   10.3
2    8.6     65   10.3
3    8.8     63   10.2
4   10.5     72   16.4
5   10.7     81   18.8
6   10.8     83   19.7
7   11.0     66   15.6
8   11.0     75   18.2
9   11.1     80   22.6
10  11.2     75   19.9
11  11.3     79   24.2
12  11.4     76   21.0
13  11.4     76   21.4
14  11.7     69   21.3
15  12.0     75   19.1
16  12.9     74   22.2
17  12.9     85   33.8
18  13.3     86   27.4
19  13.7     71   25.7
20  13.8     64   24.9
21  14.0     78   34.5
22  14.2     80   31.7
23  14.5     74   36.3
24  16.0     72   38.3
25  16.3     77   42.6
26  17.3     81   55.4
```

```
27  17.5     82   55.7
28  17.9     80   58.3
29  18.0     80   51.5
30  18.0     80   51.0
31  20.6     87   77.0
```

```r
str(trees)
```

```
'data.frame':   31 obs. of  3 variables:
 $ Girth : num  8.3 8.6 8.8 10.5 10.7 10.8 11 11 11.1 11.2 ...
 $ Height: num  70 65 63 72 81 83 66 75 80 75 ...
 $ Volume: num  10.3 10.3 10.2 16.4 18.8 19.7 15.6 18.2 22.6 19.9 ...
```

Subset a column

```r
trees$Height
```

```
 [1] 70 65 63 72 81 83 66 75 80 75 79 76 76 69 75 74 85 86 71 64 78 80 74 72 77
[26] 81 82 80 80 80 87
```

Get attributes from the data frame

```r
attributes(trees)
```

```
$names
[1] "Girth"  "Height" "Volume"

$class
[1] "data.frame"

$row.names
 [1]  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25
[26] 26 27 28 29 30 31
```

```r
names(trees)
```

```
[1] "Girth"  "Height" "Volume"
```

```r
colnames(trees)[2:3]
```

```
[1] "Height" "Volume"
```

Lists

Inevestigating data frame from before

```r
is.list(trees)
```

```
[1] TRUE
```

```r
is.data.frame(trees)
```

```
[1] TRUE
```

Can subset as a list

```r
trees[1:2]
```

```
   Girth Height
1    8.3     70
2    8.6     65
3    8.8     63
4   10.5     72
5   10.7     81
6   10.8     83
7   11.0     66
8   11.0     75
9   11.1     80
10  11.2     75
11  11.3     79
12  11.4     76
13  11.4     76
14  11.7     69
15  12.0     75
16  12.9     74
17  12.9     85
```

```
18  13.3     86
19  13.7     71
20  13.8     64
21  14.0     78
22  14.2     80
23  14.5     74
24  16.0     72
25  16.3     77
26  17.3     81
27  17.5     82
28  17.9     80
29  18.0     80
30  18.0     80
31  20.6     87
```

```
trees[[2]]
```

```
 [1] 70 65 63 72 81 83 66 75 80 75 79 76 76 69 75 74 85 86 71 64 78 80 74 72 77
[26] 81 82 80 80 80 87
```

Look at linear model fit

```
fit  <- lm(Volume ~ Height + Girth, data = trees)
```

Look at structure but restrict info (1st level of structure):

```
str(fit, max.level = 1)
```

```
List of 12
 $ coefficients : Named num [1:3] -57.988 0.339 4.708
  ..- attr(*, "names")= chr [1:3] "(Intercept)" "Height" "Girth"
 $ residuals    : Named num [1:31] 5.462 5.746 5.383 0.526 -1.069 ...
  ..- attr(*, "names")= chr [1:31] "1" "2" "3" "4" ...
 $ effects      : Named num [1:31] -167.985 53.863 69.159 -0.884 -2.007 ...
  ..- attr(*, "names")= chr [1:31] "(Intercept)" "Height" "Girth" "" ...
 $ rank         : int 3
 $ fitted.values: Named num [1:31] 4.84 4.55 4.82 15.87 19.87 ...
  ..- attr(*, "names")= chr [1:31] "1" "2" "3" "4" ...
 $ assign       : int [1:3] 0 1 2
 $ qr           :List of 5
```

```
  ..- attr(*, "class")= chr "qr"
 $ df.residual  : int 28
 $ xlevels      : Named list()
 $ call         : language lm(formula = Volume ~ Height + Girth, data = trees)
 $ terms        :Classes 'terms', 'formula'  language Volume ~ Height + Girth
  .. ..- attr(*, "variables")= language list(Volume, Height, Girth)
  .. ..- attr(*, "factors")= int [1:3, 1:2] 0 1 0 0 0 1
  .. .. ..- attr(*, "dimnames")=List of 2
  .. ..- attr(*, "term.labels")= chr [1:2] "Height" "Girth"
  .. ..- attr(*, "order")= int [1:2] 1 1
  .. ..- attr(*, "intercept")= int 1
  .. ..- attr(*, "response")= int 1
  .. ..- attr(*, ".Environment")=<environment: R_GlobalEnv>
  .. ..- attr(*, "predvars")= language list(Volume, Height, Girth)
  .. ..- attr(*, "dataClasses")= Named chr [1:3] "numeric" "numeric" "numeric"
  .. .. ..- attr(*, "names")= chr [1:3] "Volume" "Height" "Girth"
 $ model        :'data.frame':  31 obs. of  3 variables:
  ..- attr(*, "terms")=Classes 'terms', 'formula'  language Volume ~ Height + Girth
  .. .. ..- attr(*, "variables")= language list(Volume, Height, Girth)
  .. .. ..- attr(*, "factors")= int [1:3, 1:2] 0 1 0 0 0 1
  .. .. .. ..- attr(*, "dimnames")=List of 2
  .. .. ..- attr(*, "term.labels")= chr [1:2] "Height" "Girth"
  .. .. ..- attr(*, "order")= int [1:2] 1 1
  .. .. ..- attr(*, "intercept")= int 1
  .. .. ..- attr(*, "response")= int 1
  .. .. ..- attr(*, ".Environment")=<environment: R_GlobalEnv>
  .. .. ..- attr(*, "predvars")= language list(Volume, Height, Girth)
  .. .. ..- attr(*, "dataClasses")= Named chr [1:3] "numeric" "numeric" "numeric"
  .. .. .. ..- attr(*, "names")= chr [1:3] "Volume" "Height" "Girth"
 - attr(*, "class")= chr "lm"
```

Some helper functions exist

```
fit$coefficients
```

```
(Intercept)      Height       Girth
-57.9876589   0.3392512   4.7081605
```

```
coef(fit)
```

```
(Intercept)      Height       Girth
-57.9876589   0.3392512   4.7081605
```

```r
fit$residuals
```

```
          1           2           3           4           5           6 
 5.46234035  5.74614837  5.38301873  0.52588477 -1.06900844 -1.31832696 
          7           8           9          10          11          12 
-0.59268807 -1.04594918  1.18697860 -0.28758128  2.18459773 -0.46846462 
         13          14          15          16          17          18 
-0.06846462  0.79384587 -4.85410969 -5.65220290  2.21603352 -6.40648192 
         19          20          21          22          23          24 
-4.90097760 -3.79703501  0.11181561 -4.30831896  0.91474029 -3.46899800 
         25          26          27          28          29          30 
-2.27770232  4.45713224  3.47624891  4.87148717 -2.39932888 -2.89932888 
         31 
 8.48469518 
```

```r
residuals(fit)
```

```
          1           2           3           4           5           6 
 5.46234035  5.74614837  5.38301873  0.52588477 -1.06900844 -1.31832696 
          7           8           9          10          11          12 
-0.59268807 -1.04594918  1.18697860 -0.28758128  2.18459773 -0.46846462 
         13          14          15          16          17          18 
-0.06846462  0.79384587 -4.85410969 -5.65220290  2.21603352 -6.40648192 
         19          20          21          22          23          24 
-4.90097760 -3.79703501  0.11181561 -4.30831896  0.91474029 -3.46899800 
         25          26          27          28          29          30 
-2.27770232  4.45713224  3.47624891  4.87148717 -2.39932888 -2.89932888 
         31 
 8.48469518 
```

if/then/else

Fizz buzz challenge

- take in a number
- if it is divisible by 3 return fizz
- if it is divisible by 5 return buzz
- if it is divisible by 15 return fizz buzz

```r
number <- 15
if ((number %% 15) == 0) {
  print ("fizz buzz")
} else if ((number %% 5) == 0) {
  print ("buzz")
} else if ((number %% 3) == 0) {
  print ("fizz")
} else {
  print ("whoops?")
}
```

```
[1] "fizz buzz"
```

```r
#if (!(number %% 15))
```

Loops

Wrap the fizz buzz code into a loop to check for multiple values.

```r
for (number in -1:41){
  if ((number %% 15) == 0) {
    print ("fizz buzz")
  } else if ((number %% 5) == 0) {
    print ("buzz")
  } else if ((number %% 3) == 0) {
    print ("fizz")
  } else {
    print ("whoops?")
  }
}
```

```
[1] "whoops?"
[1] "fizz buzz"
[1] "whoops?"
[1] "whoops?"
[1] "fizz"
[1] "whoops?"
[1] "buzz"
[1] "fizz"
```

```
[1] "whoops?"
[1] "whoops?"
[1] "fizz"
[1] "buzz"
[1] "whoops?"
[1] "fizz"
[1] "whoops?"
[1] "whoops?"
[1] "fizz buzz"
[1] "whoops?"
[1] "whoops?"
[1] "fizz"
[1] "whoops?"
[1] "buzz"
[1] "fizz"
[1] "whoops?"
[1] "whoops?"
[1] "fizz"
[1] "buzz"
[1] "whoops?"
[1] "fizz"
[1] "whoops?"
[1] "whoops?"
[1] "fizz buzz"
[1] "whoops?"
[1] "whoops?"
[1] "fizz"
[1] "whoops?"
[1] "buzz"
[1] "fizz"
[1] "whoops?"
[1] "whoops?"
[1] "fizz"
[1] "buzz"
[1] "whoops?"
```

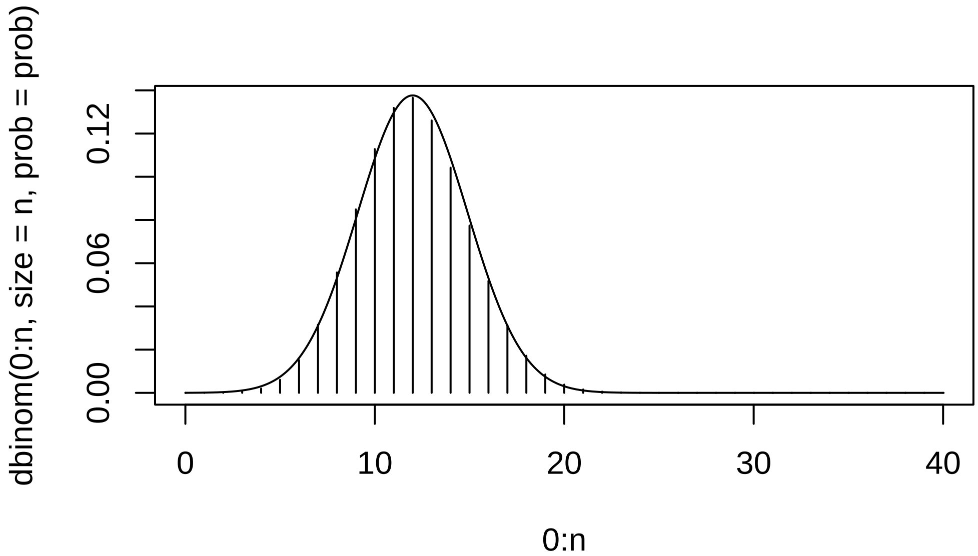
Writing R Functions

Normal (continuous) Approximation to the binomial (discrete).

```r
n <- 40
prob <- 0.3
#probabilities from a binomial RV
dbinom(0:n, size = n, prob = prob)
```

```
 [1] 6.366806e-07 1.091452e-05 9.121424e-05 4.951630e-04 1.962968e-03
 [6] 6.057157e-03 1.514289e-02 3.152194e-02 5.572629e-02 8.491625e-02
[11] 1.128173e-01 1.318644e-01 1.365738e-01 1.260681e-01 1.041992e-01
[16] 7.740510e-02 5.183378e-02 3.136161e-02 1.717422e-02 8.522543e-03
[21] 3.835144e-03 1.565365e-03 5.793884e-04 1.943290e-04 5.899274e-05
[26] 1.618087e-05 4.000763e-06 8.890585e-07 1.769045e-07 3.137223e-08
[31] 4.929921e-09 6.815560e-10 8.215184e-11 8.535256e-12 7.531108e-13
[36] 5.533059e-14 3.293487e-15 1.525940e-16 5.162955e-18 1.134715e-19
[41] 1.215767e-21
```

```r
#plot with plot
plot(0:n,
     dbinom(0:n, size = n, prob = prob),
     type = "h"
)
norm_x <- seq(from = 0, to = n, length = 1000)
lines(norm_x,
      dnorm(norm_x, mean = n*prob, sd = sqrt(n*prob*(1-prob))))
```

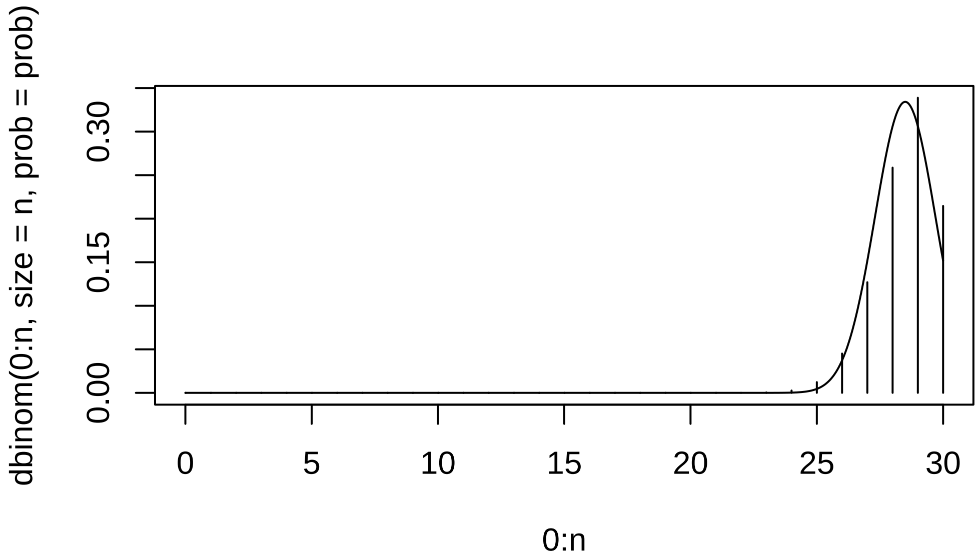


Let's write a function to make this plot for any n and p we give it.

```
plot_norm_approx <- function (n, prob) {
  #plot with plot
  plot(0:n,
       dbinom(0:n, size = n, prob = prob),
       type = "h"
  )
  norm_x <- seq(from = 0, to = n, length = 1000)
  lines(norm_x,
        dnorm(norm_x, mean = n*prob, sd = sqrt(n*prob*(1-prob))))
}
```

Test it.

```
plot_norm_approx(30, 0.95)
```

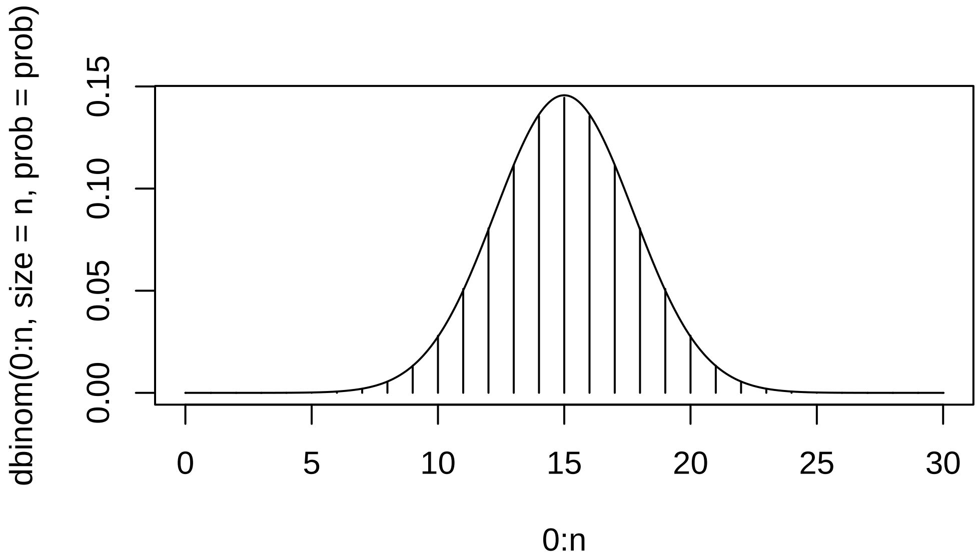


Add some default values.

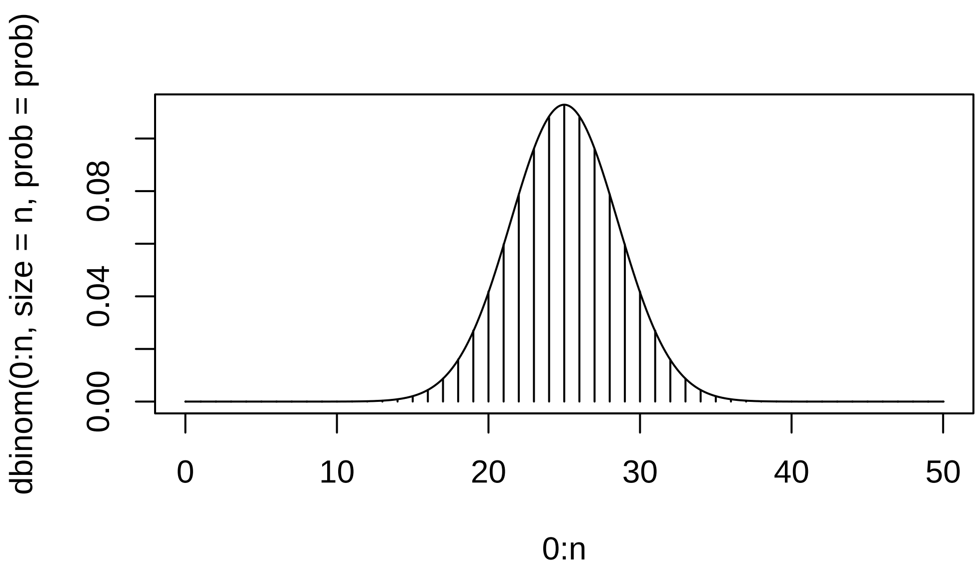
```
plot_norm_approx <- function (n = 30, prob = 0.5) {
  #plot with plot
  plot(0:n,
       dbinom(0:n, size = n, prob = prob),
       type = "h"
  )
  norm_x <- seq(from = 0, to = n, length = 1000)
  lines(norm_x,
        dnorm(norm_x, mean = n*prob, sd = sqrt(n*prob*(1-prob))))
}
```

Test it.

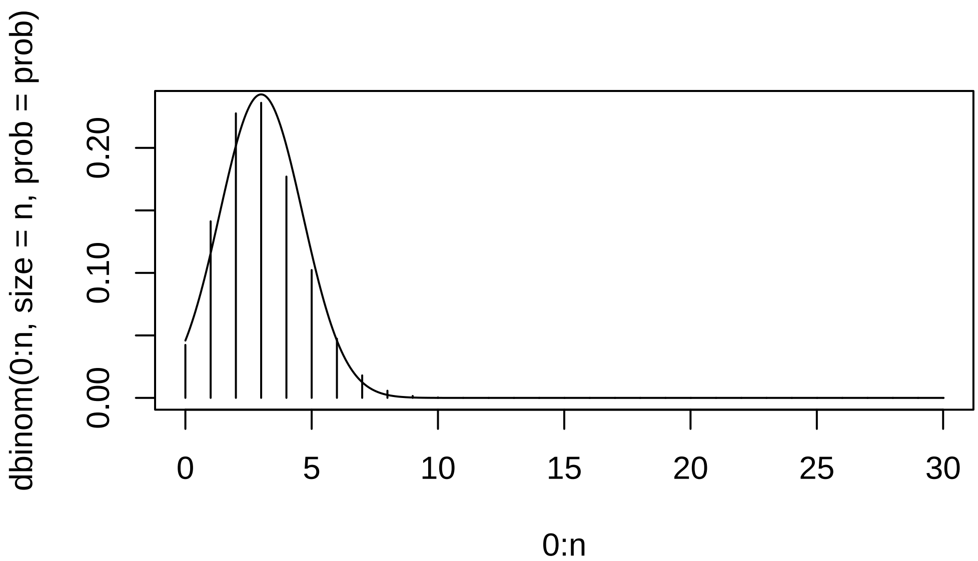
```
#default
plot_norm_approx()
```



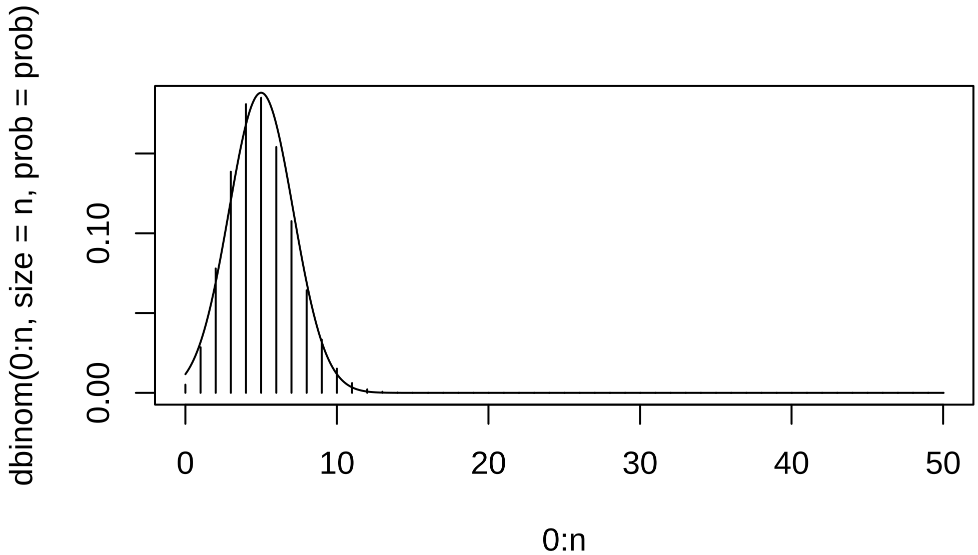

```r
#change n to be 50
plot_norm_approx(50)
```



```
#change prob to be 0.1
plot_norm_approx(prob = 0.1)
```




```
#call things positionaly or call them by name
plot_norm_approx(prob = 0.1, 50)
```



dbinom(0:n, size = n, prob = prob)
0.00
0.10
0
10
20
30
40
50
0:n