BTRY4830_Lab3

Kate Harline

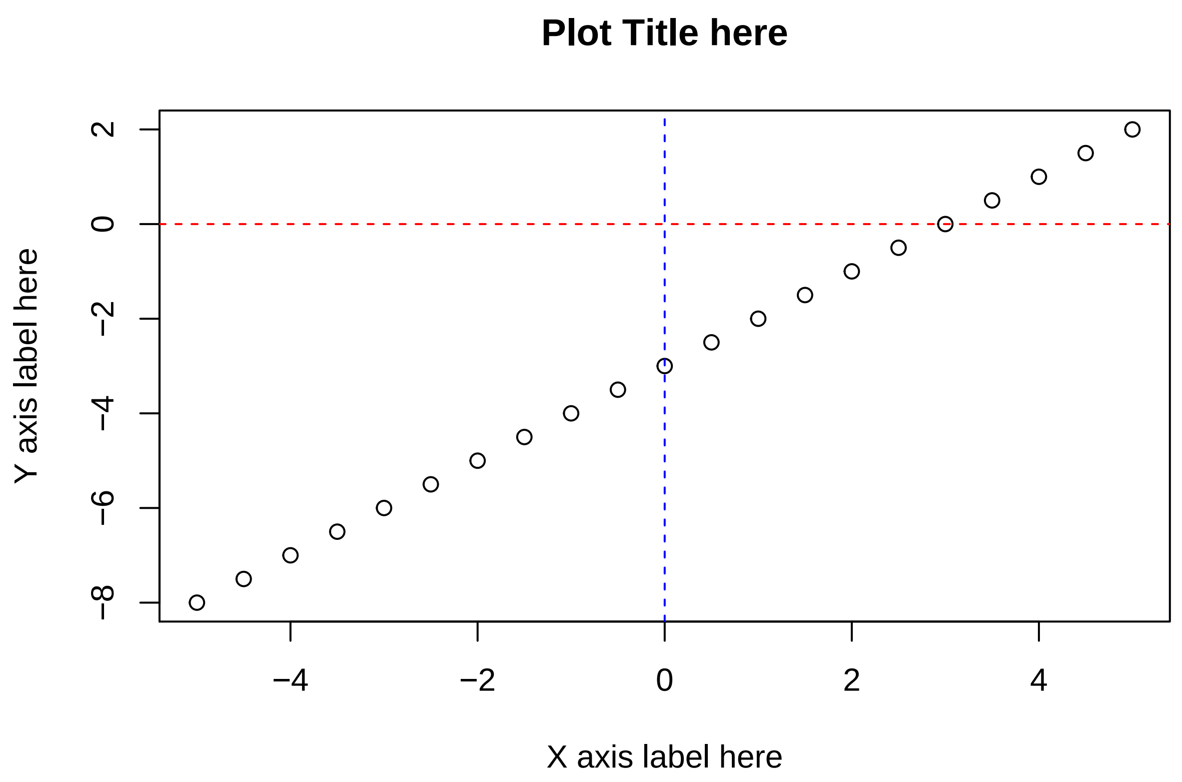
2/14/2019

1. Basic Plotting Using R

1.1 Plot some data

For this class, you will want to visualize some of your data. R has some built-in plotting functions like **plot()** which generates an X-Y plot. The basic syntax is **plot(x_axis, y_axis)**. We can add more information to the plot by passing optional axis labels and titles to the plot function.

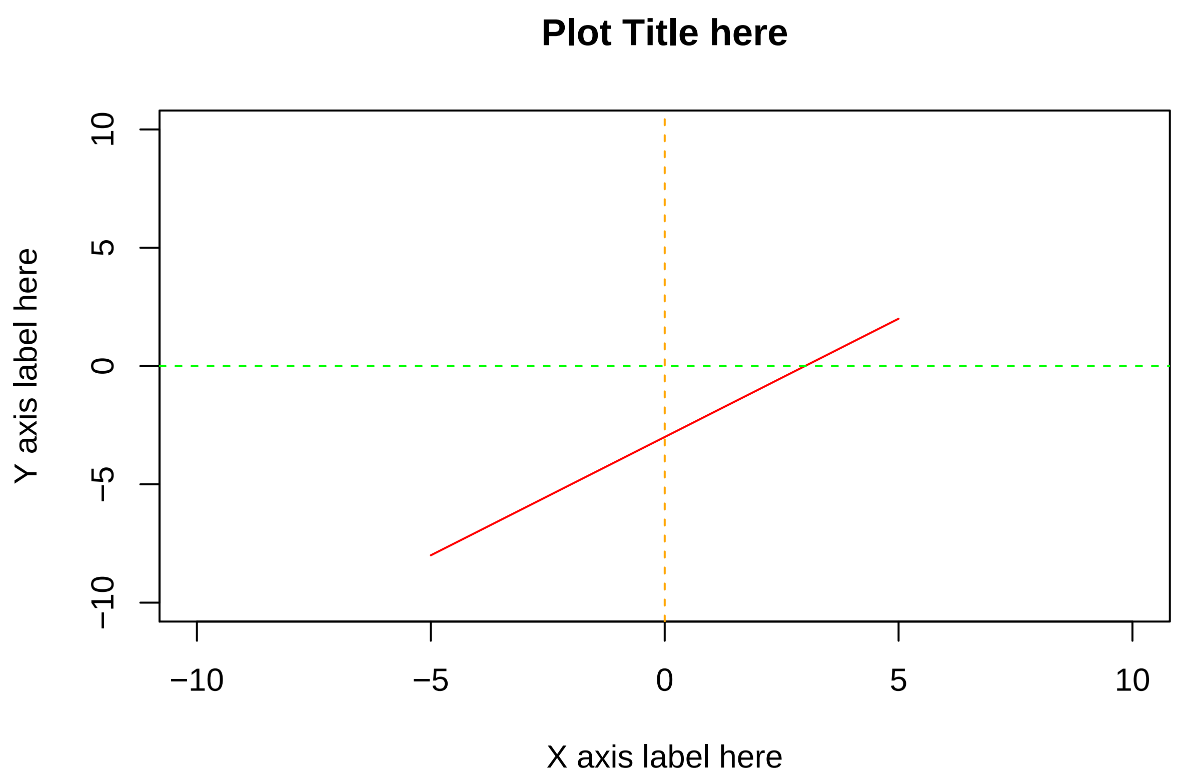
```r
# Assign values to the variables you want to plot
x <- seq(-5, 5, by = 0.5)     # generate a vector of numbers from -5 to 5 incremented by 0.5
y <- x -3                     # element-wise subtract 3 from each value in the x vector

# Format and plot figure
plot(x, y,
     xlab = 'X axis label here',
     ylab = 'Y axis label here',
     main = 'Plot Title here')

## The function abline adds one or more straight lines through the CURRENT plot
abline(h = 0, lty = 2, col = 'red')
abline(v = 0, lty = 2, col = 'blue')
```



```
# Type ?abline into the console for more information about this function
```

What if we wanted to make a line plot instead of a point plot like the one above? There are different plot options that you can read about by typing **?plot** into the console and looking through the *type* parameter section.

```
# Format and plot figure (line)
plot(x, y,                                    # data
     type = 'l',                              # specifying type, 'p' for points, 'l' for lines, ...
     xlab = 'X axis label here',              # x axis label
     ylab = 'Y axis label here',              # y axis label
     main = 'Plot Title here',                # title
     xlim = c(-10,10),                        # set range of x
     ylim = c(-10,10),                        # set range of x
     col = 'red')                             # set color of data

# col can be a color name defined in R or a color generated by the function rgb()
# or a code for a color.
abline(h = 0, lty = 2, col = rgb(0.0, 1.0, 0.0))
abline(v = 0, lty = 2, col = 'orange')
```

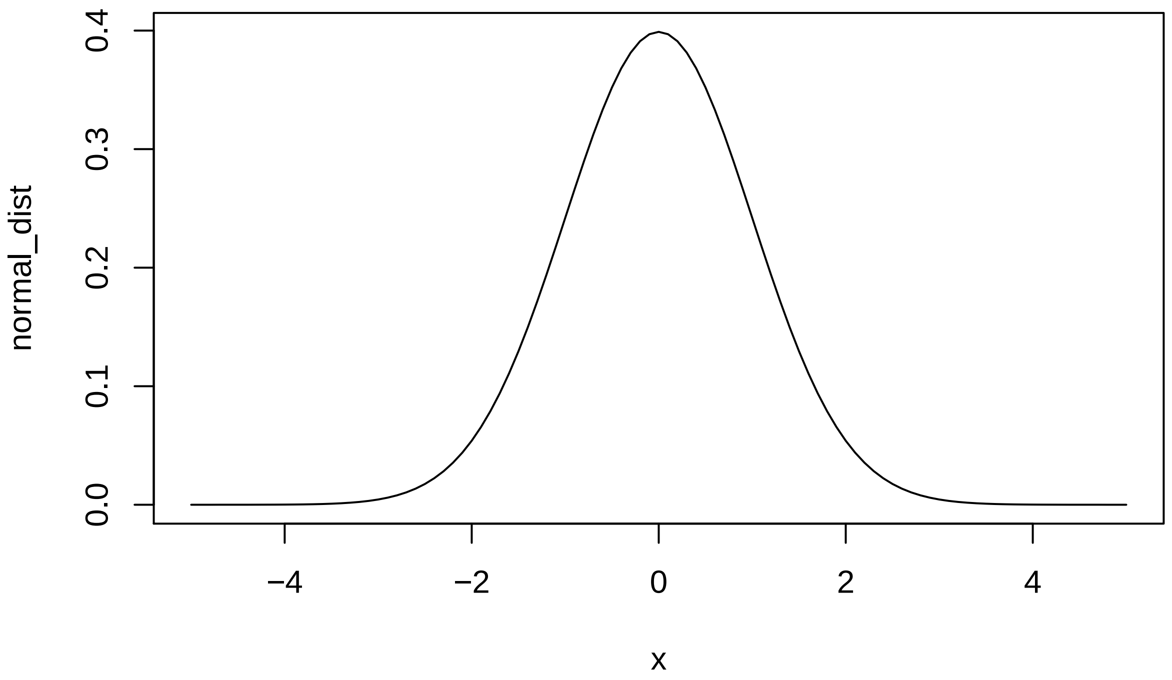


```
# google R colors or see http://www.stat.columbia.edu/~tzheng/files/Rcolor.pdf
# for a summary of colors you can use
```

1.2 Plot functions

You can also plot functions directly with **plot()**

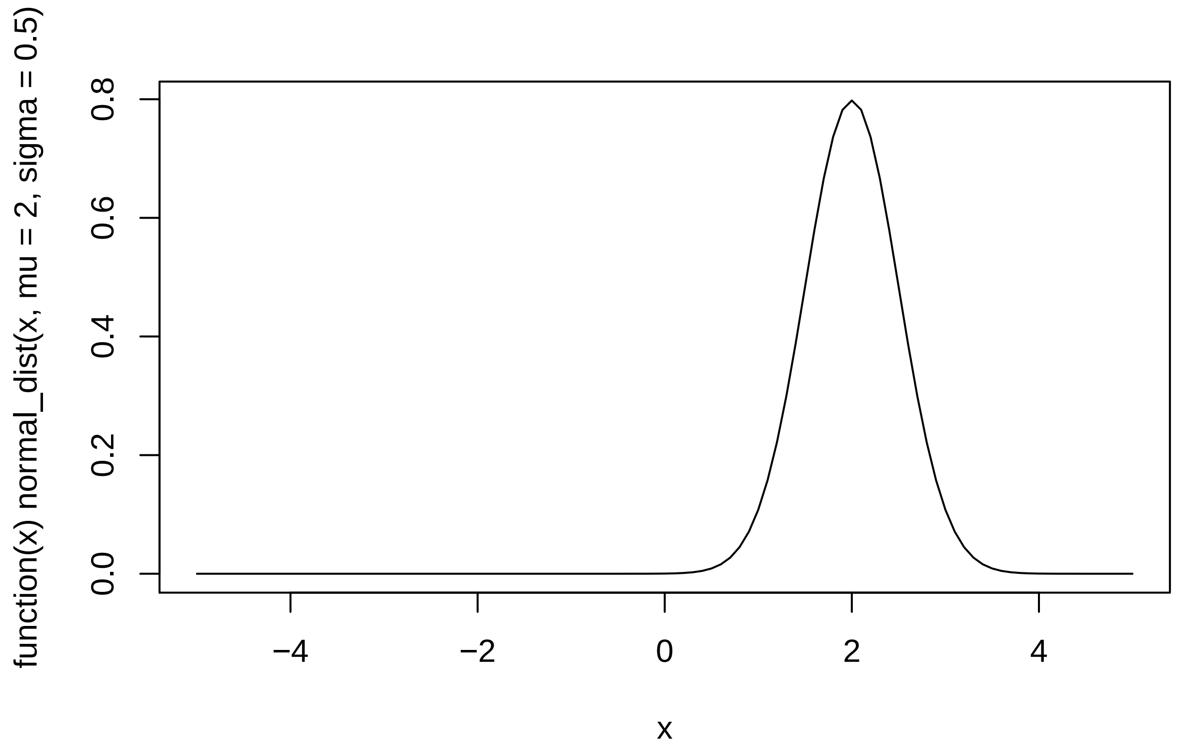
```
normal_dist <- function(x, mu = 0, sigma = 1) {
  1/ (sigma * sqrt(2 * pi)) * exp( - ((x-mu)^2) / (2 * sigma ^ 2) )
}

plot(normal_dist, xlim = c(-5,5))
```



```
# if you don't specify xlim, the default will be from 0 to 1
```

Notice, that for the above function, we've essentially set default values for μ and σ. If we want to change those values, we can do as follows.

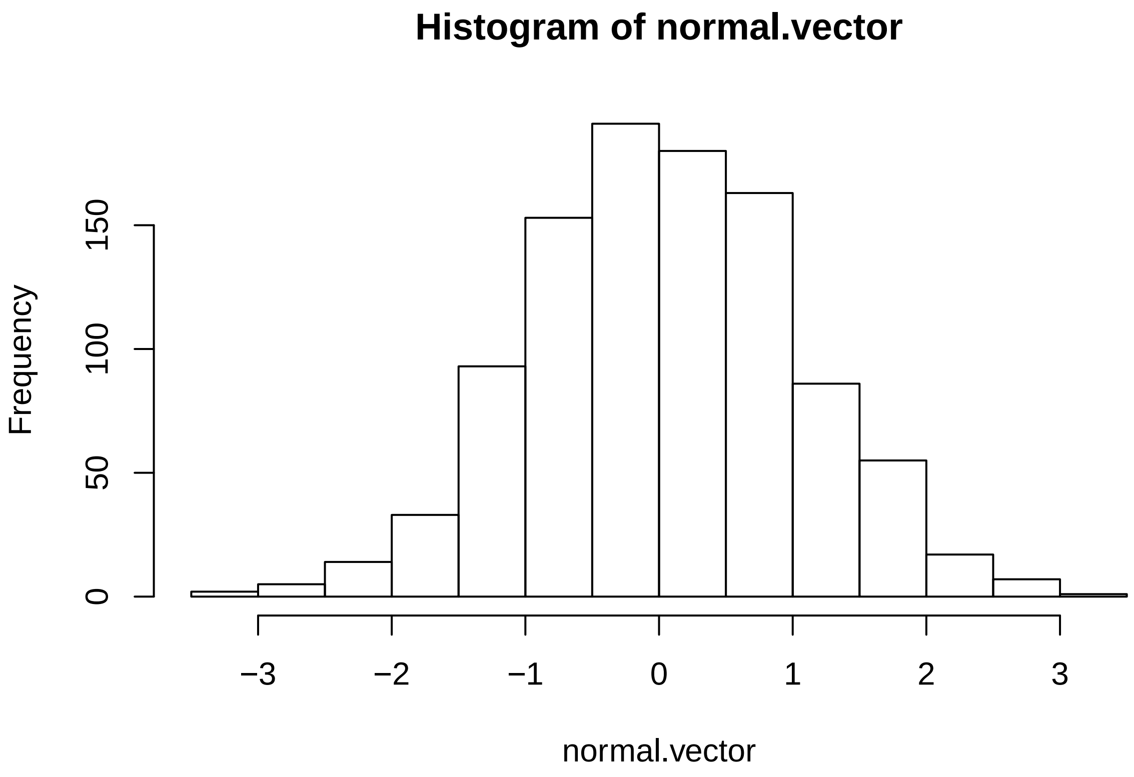
```
plot( function(x) normal_dist(x, mu = 2, sigma = 0.5), xlim = c(-5,5))
```



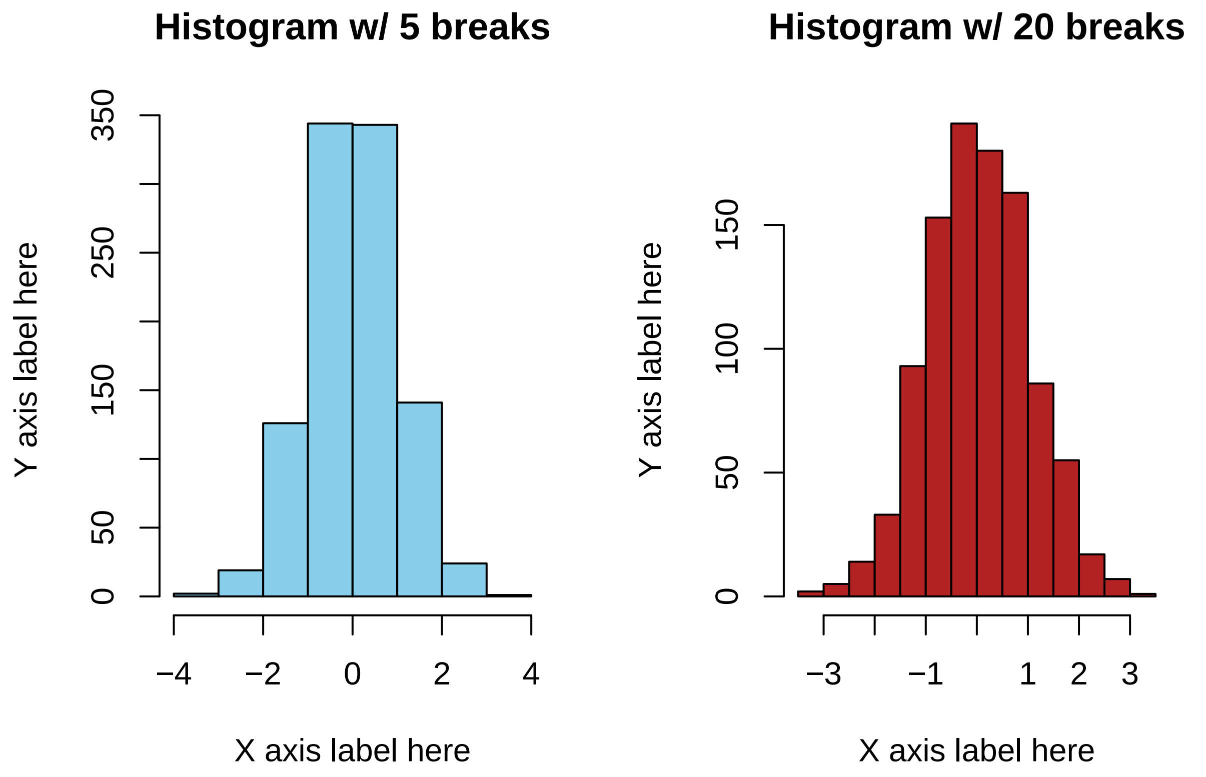
1.3 Plot histograms/Multi-plot figures

We can also generate histograms using the **hist()** function, which often allows us a good opportunity to inspect the initial data.

```r
normal.vector <- rnorm(1000)

# rnorm() generates random points froma  normal distribution
# more information about this function will be covered below.

# most basic form
hist(normal.vector)
```



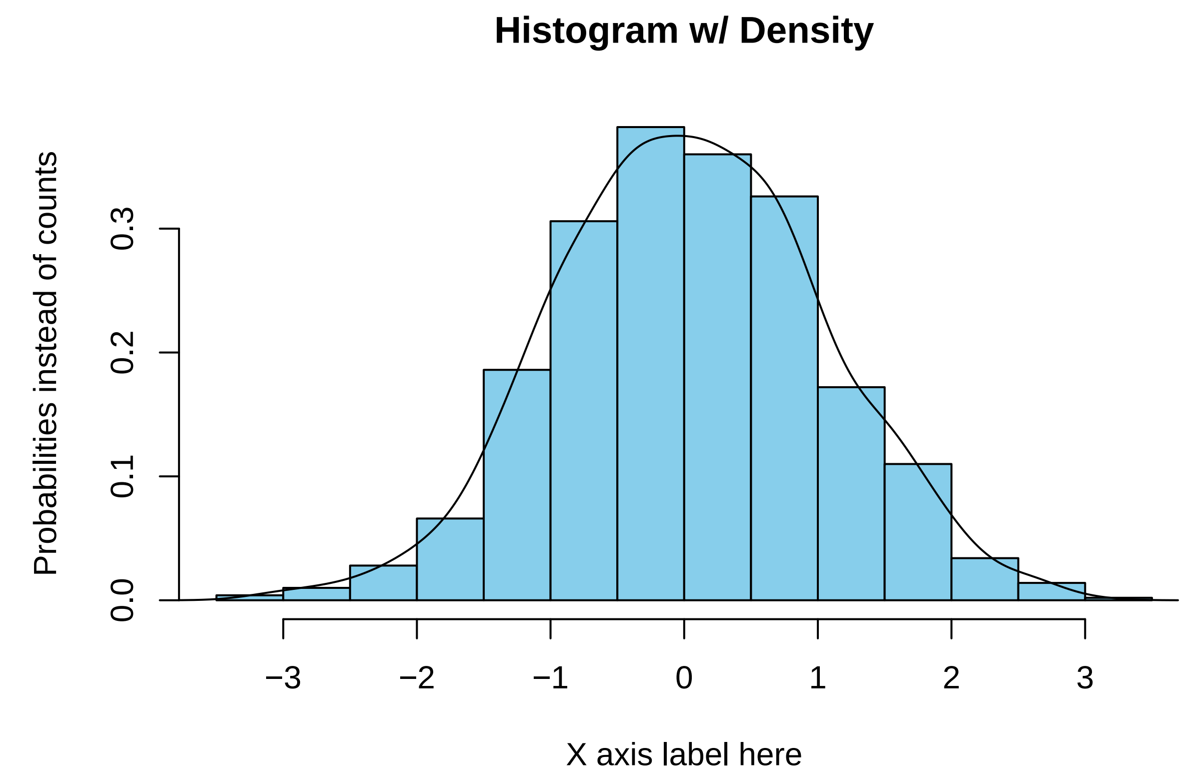
```r
# adding options

### par() can be used to set or query graphical parameters.
### mfrow=c(nr,nc) essentially says that subsequent figures will be
### drawn in an nr-by-nc array on the device

par(mfrow = c(1,2))

# most basic form
hist(normal.vector,                        # data
     xlab = 'X axis label here',           # x axis label
     ylab = 'Y axis label here',           # y axis label
     main = 'Histogram w/ 5 breaks',       # title
     col = 'skyblue',                      # set fill color for bars
     breaks = 5)                           # breaks specifies number of bins (kind of)

hist(normal.vector,                        # data
     xlab = 'X axis label here',           # x axis label
     ylab = 'Y axis label here',          # y axis label
     main = 'Histogram w/ 20 breaks',     # title
     col = 'firebrick',                   # set fill color for bars
     breaks = 20)                         # breaks specifies number of bins (kind of)
```



```
# Certain inputs for the break parameter are treated as suggestions like integers so the
# number of breaks you specify won't necessarily be the exact number of bins the histogram
# is broken up into. Breaks are explained further here:
# https://stackoverflow.com/questions/49438936/understanding-hist-and-break-intervals-in-r


# par(mfrow=c(1,1)) What happens if we don't reset par?

# You can set specific bins by giving "breaks" a vector
# hist(normal.vector,                                 # data
#      xlab = 'X axis label here',                    # x axis label
#      ylab = 'Y axis label here',                    # y axis label
#      main = 'Histogram w/ [-4,4] by 0.5 bins',      # title
#      col = 'firebrick',                             # set fill color for bars
#      breaks = seq(-4,4,0.5))                        # breaks specifies number of bins (kind of)
```

If we set the option probability to TRUE, we will change the y axis from counts to probability densities. We can also add a density curve using **lines()** with **density()**. **density()** computes kernel density estimates (essentially a non-parametric way of estimating the probability density function of a random variable)

```
hist(normal.vector,                                 # data
     xlab = 'X axis label here',                    # x axis label
     ylab = 'Probabilities instead of counts',      # y axis label
     main = 'Histogram w/ Density',                 # title
     col = 'skyblue',                               # set fill color for bars
     probability = TRUE)                                # density instead of counts

lines(density(normal.vector))
```



1.4 Save plots

We can also save plots directly to a file. We can either use RStudio or code it up ourselves.

```
# the png function creates a png file at a custom location.
# you can also specify the resolution and the size of the image, look up ?png for details
# Remember to check/set your working directory! (getwd and setwd functions)
png(file = './Normal_distribution_histograms.png')

hist(normal.vector,
    xlab = 'X label',
    main = 'Plot saved on disk',
    col = 'skyblue',
    probability = TRUE)

lines(density(normal.vector))

dev.off()

# After plotting on the png file, the png file has to be closed in order to save the plot
# correctly. Otherwise, subsequent plots will be written to the same png file and that
# means trouble.
```

2. Plotting using ggplot

Up to this point, we've been using R's default plotters. You can also use R packages specifically designed for generating plots. By far, the most popular package is ggplot (technically ggplot2) which is highly flexible and makes attractive plots. It needs a little more work than basic R plotting but it has some advantages that many people prefer.

Some Resources:
-General Intro–https://ggplot2.tidyverse.org
-Cheatsheet–https://www.rstudio.com/wp-content/uploads/2015/03/ggplot2-cheatsheet.pdf

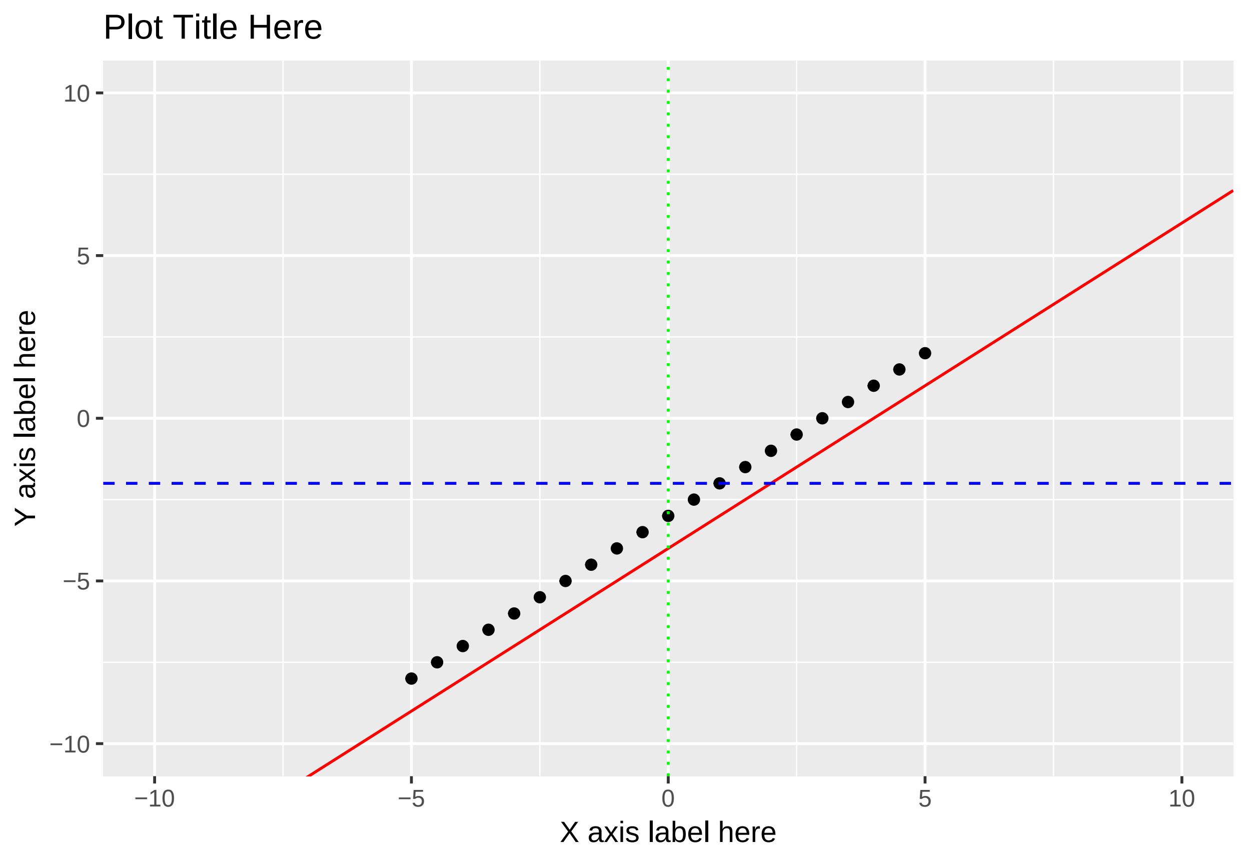
2.1 Plot some data

For our purposes, we start visualization with the function, **ggplot**, which needs to be supplied with a dataframe and an aesthetic mapping using **aes()** (the aesthetic mapping indicates which columns of data are plotted on which axis). We can add on layers to get exactly the kind of plot we want

```r
#install.packages("ggplot2") if you need to install ggplot2 still
library(ggplot2)
```

```
## Warning: package 'ggplot2' was built under R version 3.4.4
```

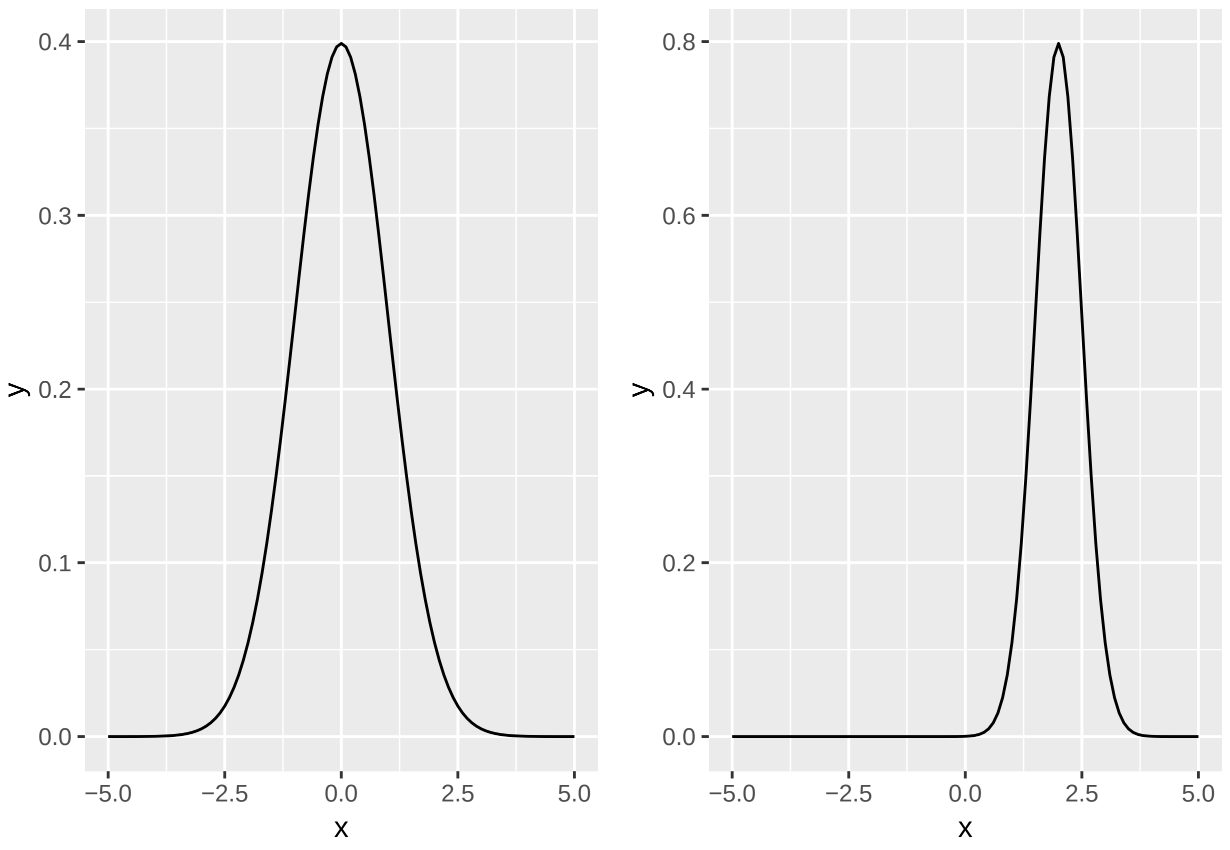
```r
# The first thing we need to do is turn our values into a dataframe

plotting_dataframe <- data.frame(x_vals=x, y_vals=y)           # build a dataframe from x and y initia

ggplot(plotting_dataframe, aes(x_vals, y_vals)) +              # data and aesthetic mapping
  geom_point() +                                               # the geometric markings we want, here p
  labs(title='Plot Title Here',                                # List all of the labels you want
       x='X axis label here',
       y='Y axis label here') +
  geom_abline(intercept=-4, slope=1, col='red') +              # Generic function to add line to plot
  geom_hline(yintercept=-2, col='blue', linetype='dashed') +   # Add horizontal line
  geom_vline(xintercept=0, col='green', linetype='dotted') +   # Add vertical line
  xlim(-10,10) + ylim(-10,10)                                  # Ranges for x and y
```



2.2 Plot functions/Multi-plot figures

If we want to plot multiple plots on one figure, we should use the **gridExtra** package to arrange them. We'll do this in the next code chunk.

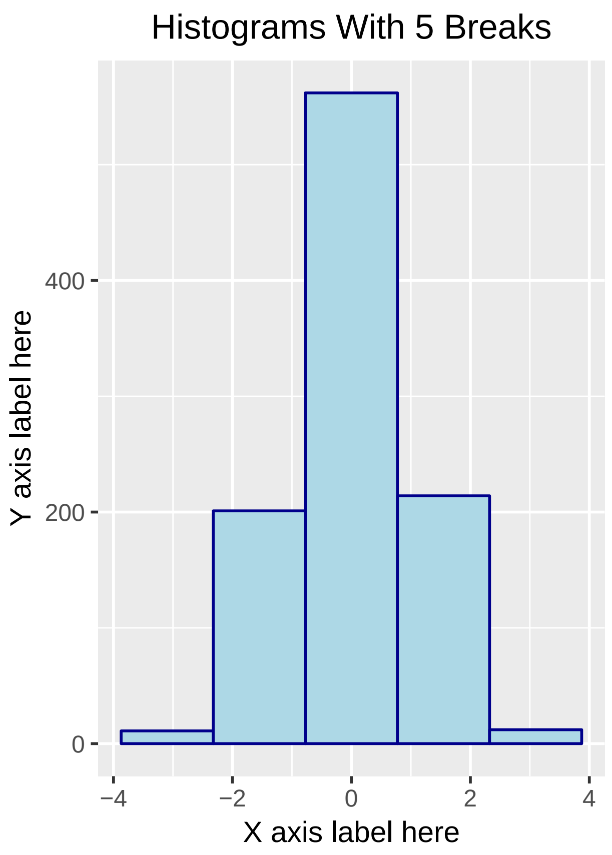
```
#install.packages("gridExtra")
library(gridExtra)

# plot function we defined above (normal_dist) over the specified x range
# ggplot returns ggplot object which is stored into the plot1 variable
plot1 <- ggplot(data.frame(x=c(-5,5)),aes(x)) +
            stat_function(fun=normal_dist)

# example of how to plot function while specifying other arguments
plot2 <- ggplot(data.frame(x = c(-5,5)), aes(x)) +
            stat_function(fun=normal_dist, args = list(mu=2, sigma=0.5))

# analagous to par function shown above for building figures with multiple pltos
grid.arrange(plot1,plot2, nrow=1, ncol=2)
```

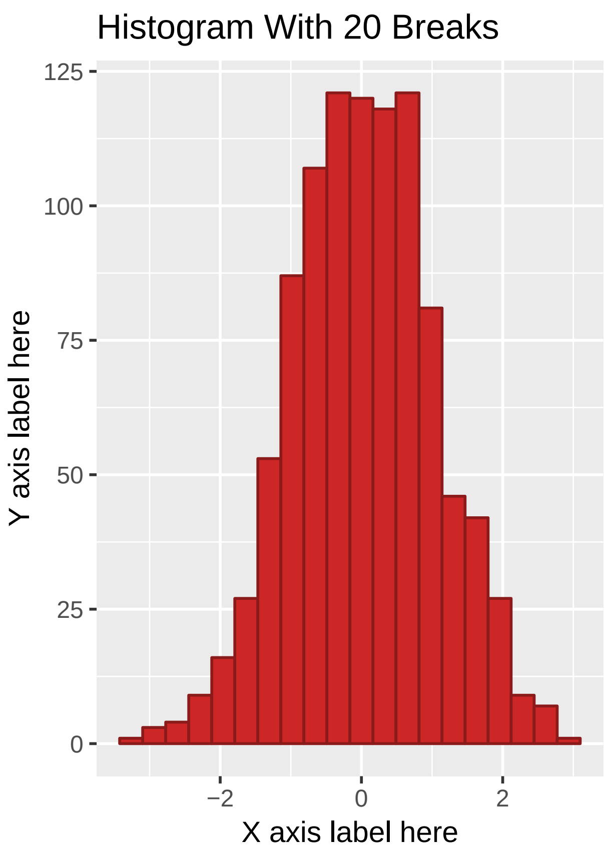


2.3 Plot histograms

We can also generate histograms using ggplot.

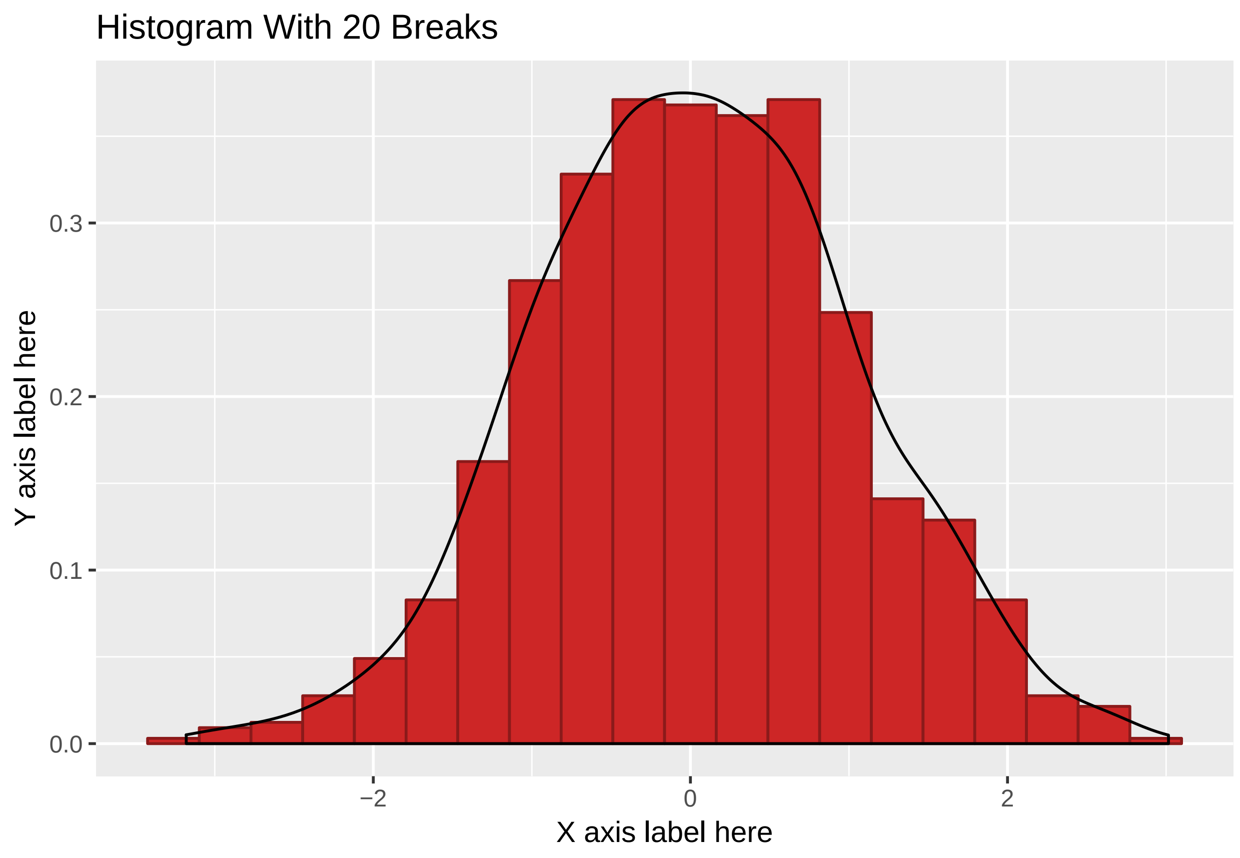
```
plotting_histogram <- data.frame(normal.vector <- normal.vector)

hist1 <- ggplot(plotting_histogram, aes(normal.vector)) +
  geom_histogram(bins=5, col='darkblue', fill='lightblue') +
  labs(title='Histograms With 5 Breaks', x='X axis label here', y='Y axis label here') +
  theme(plot.title=element_text(hjust = 0.5))

hist2 <- ggplot(plotting_histogram, aes(normal.vector)) +
  geom_histogram(bins=20, col='firebrick4', fill='firebrick3') +
  labs(title='Histogram With 20 Breaks', x='X axis label here', y='Y axis label here')

grid.arrange(hist1, hist2, nrow=1, ncol=2)
```





. . . and we can plot densities as we did above with the built-in R functions:

```
ggplot(plotting_histogram, aes(normal.vector)) +
  geom_histogram(aes(y=..density..),
                 bins=20,
                 col='firebrick4',
                 fill='firebrick3') +
  geom_density(colour='black') +
  labs(title='Histogram With 20 Breaks',
       x='X axis label here',
       y='Y axis label here')
```



2.4 Save plots

Saving using ggplot is a little different than the built-in R functions. The library includes a function for saving the plots called **ggsave()**. Since ggplot returns a plotting object, you can specify the plot object you want to save within a single line anywhere in the script after the plot was created.

```
plot2save <- ggplot(plotting_histogram, aes(normal.vector)) +
  geom_histogram(aes(y=..density..),
               bins=20,
               col='firebrick4',
               fill='firebrick3') +
  geom_density(colour='black') +
  labs(title='Histogram With 20 Breaks',
       x='X axis label here',
       y='Y axis label here')

# Save this plot
ggsave('./savethisggplot.png', plot2save, device='png')

# Given the following information about the ggsave functiona and its parameters,
#       ggsave(filename, plot = last_plot(), device = NULL, ...) {...}
# can you figure out which lines of code will actually run through and how they might be
# different from the line of code above?

# ggsave('savethisggplot')
# ggsave('./savethisggplot.png', hist1, device='png')
```

```r
# ggsave('savethisggplot.png', plot2save, device='pdf')
# ggsave('savethisggplot.pdf')
```

When working with plots, especially the ggplot functions, think about what your expected inputs and outputs should look like and how they interact with different programming structures.

Some scope experiments you can try answering: If I plotted a figure in a script and then ran it from the console, will it pop up? How about if I stuck it in a loop or function? If it doesn't, how can I get it to pop up? If it does, how can I keep it from popping up?

3. Functions for Normal Distributions

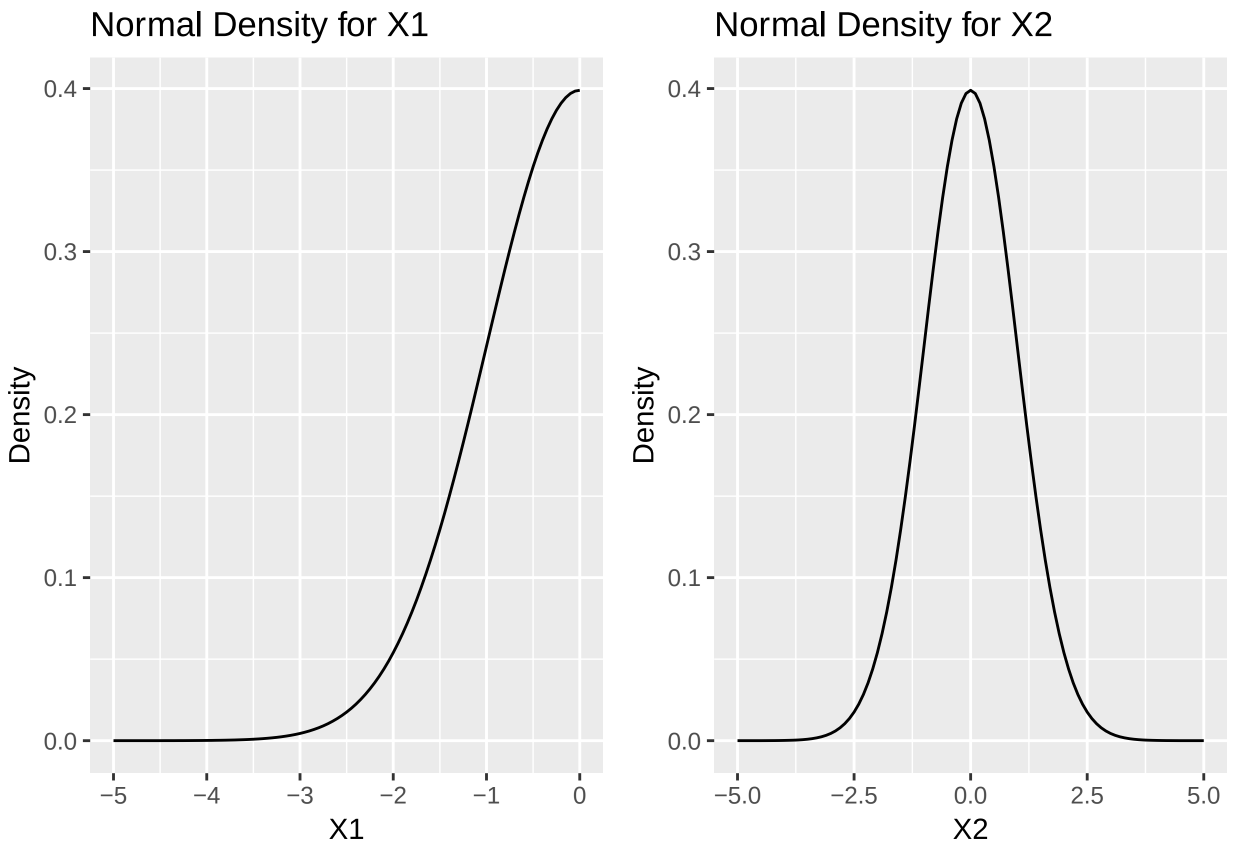
There are built-in functions for many well-studied distributions. Let's work through some of the functions for the normal distribution.

```r
# rnorm(): Generate random values sampled from a normal distribution with
# a specified mean and standard deviation

# Generate 5 values for a sample drawn from a normal distribution with mean=0
# and sd=1. This allows us to generate a sample
rnorm(n=100, mean=0, sd=1)
```

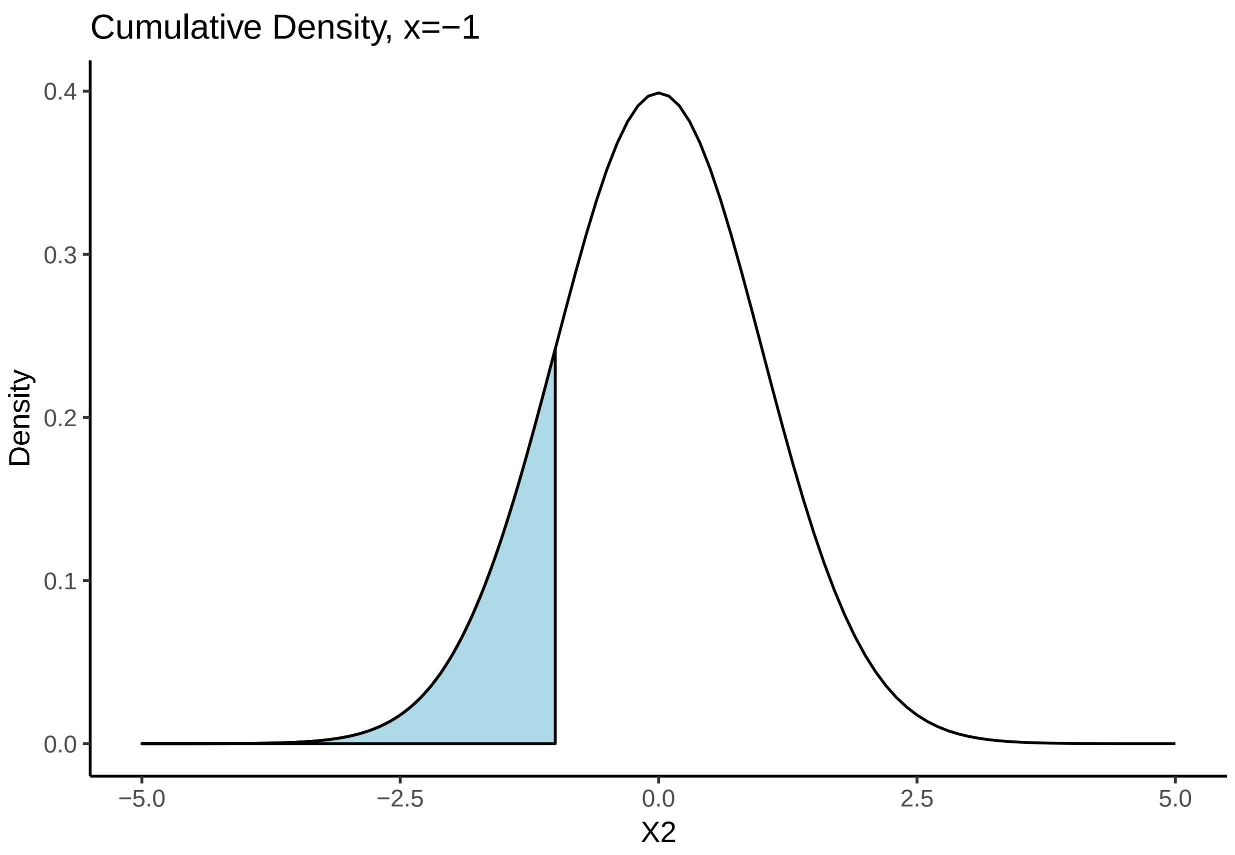
```
##   [1]  8.196961e-01  7.828116e-05  6.978480e-01  4.605013e-01 -2.364782e+00
##   [6]  2.869854e-01  2.839719e-01 -8.359656e-01  6.012629e-01 -2.943976e+00
##  [11]  7.927600e-01 -5.253346e-01 -9.358594e-01  3.533105e-01  1.047245e+00
##  [16]  5.444661e-01 -5.148584e-01 -3.799472e-01 -3.293891e-01 -2.811622e-01
##  [21]  7.280525e-01  3.789342e-01 -2.456267e-01  6.180134e-01 -2.047121e+00
##  [26] -1.218454e-01  3.689851e-01 -9.382825e-01  2.846456e-01  1.282803e+00
##  [31] -4.012020e-01 -6.680202e-01 -8.672685e-01  7.021581e-01  1.616143e+00
##  [36] -9.318024e-01  4.153637e-01  1.966611e-01  2.863906e-01 -7.212541e-01
##  [41] -3.718385e-01 -9.819201e-01  3.216909e-02 -3.981407e-01 -3.530358e-01
##  [46]  7.465248e-01  1.238536e-01  4.034929e-01  1.670818e+00 -1.937627e+00
##  [51] -6.239237e-01  4.000725e-01 -1.111403e-02  1.257289e+00  3.790028e-01
##  [56] -3.693903e-01 -1.659813e+00 -1.318681e+00 -6.104282e-01  3.494469e-01
##  [61]  3.742845e-01 -3.503619e-01  4.483189e-02  1.035621e+00 -3.560340e-01
##  [66] -3.777127e-01  1.924711e+00  2.711710e-01 -2.255278e+00  1.575087e-01
##  [71]  2.318825e+00 -9.027600e-01  4.890329e-01 -1.952628e-01 -1.771889e+00
##  [76]  8.191935e-01 -1.224501e-01  6.537117e-01  1.861554e+00  1.256690e-01
##  [81] -1.444635e+00  7.633875e-01  1.837099e-02 -4.565398e-01  1.430666e+00
##  [86] -9.954460e-01 -1.159836e+00  4.316822e-01  1.363922e+00 -3.022977e-01
##  [91] -1.583958e-01 -6.517388e-01  9.591611e-01  6.944411e-01 -6.537968e-01
##  [96] -3.814179e-01 -1.087972e+00  6.569902e-01 -3.087003e-01 -1.093262e+00
```

```r
# dnorm(): Calculates the height of the probability density function at a
# specific point
# Can plot the density function for a given interval with dnorm()

plot3 <- ggplot(data.frame(x=seq(-5,0,by=0.1)), aes(x)) +
  stat_function(fun=dnorm, args=alist(mean=0, sd=1)) +
  labs(title='Normal Density for X1', x='X1', y='Density')

plot4 <- ggplot(data.frame(x=seq(-5,5,by=0.1)), aes(x)) +
  stat_function(fun=dnorm, args=list(mean=0, sd=1)) +
  labs(title='Normal Density for X2', x='X2', y='Density')
```

```r
grid.arrange(plot3, plot4, nrow=1, ncol=2)
```



```r
# pnorm(): returns the value for the cumulative density function for a given point
# We can think of it as the area under the curve up to a certain value
# for a probability density function

# use dnorm to generate a plot with highlighted area under curve
ggplot(data.frame(x=seq(-5,5,by=0.1)), aes(x)) +
  stat_function(  fun=dnorm, args=list(mean=0, sd=1)  ) +
  stat_function(  fun=dnorm, args=list(mean=0, sd=1), xlim=c(-5,-1),
                 geom='area', col='black', fill='lightblue'  ) +
  labs(  title='Cumulative Density, x=-1', x='X2', y='Density'  ) +
  theme(  panel.grid.major = element_blank(),
          panel.grid.minor = element_blank(),
          panel.background = element_blank(),
          axis.line = element_line(colour='black')  )
```

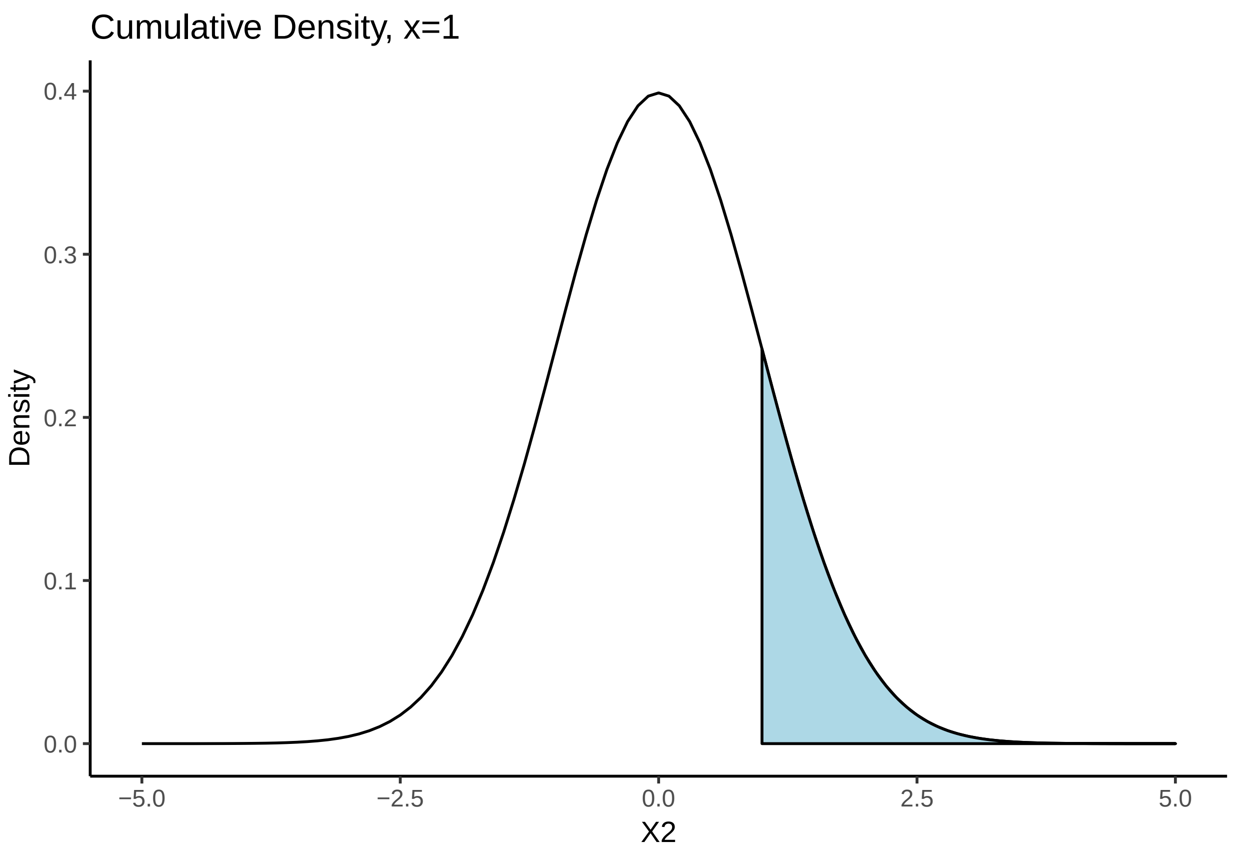


```r
# use pnorm to calculate that area
pnorm(-1, mean=0, sd=1)
```

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

We can also calculate the area from a given point to infinity by setting the *lower.tail* option to FALSE. Since the nromal distribution is symmetric around 0, this should give you the same value as *pnorm(-1)*.

```r
# use dnorm to generate a plot with highlighted area under curve
ggplot(data.frame(x=seq(-5,5,by=0.1)), aes(x)) +
  stat_function(  fun=dnorm, args=list(mean=0, sd=1)  ) +
  stat_function(  fun=dnorm, args=list(mean=0, sd=1), xlim=c(1,5),
                 geom='area', col='black', fill='lightblue'  ) +
  labs(  title='Cumulative Density, x=1', x='X2', y='Density'  ) +
  theme(  panel.grid.major = element_blank(),
          panel.grid.minor = element_blank(),
          panel.background = element_blank(),
          axis.line = element_line(colour='black')  )
```



```r
# use pnorm to calculate that area
pnorm(1, mean=0, sd=1, lower.tail=FALSE)
```

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

The functions are not limited to normal distributions. There are also functions for the *uniform*, *binomial*, *poisson*, *F*, . . . distributions. (See **rbinom()**, **rf**, **rlnorm**, . . . , **dbinom**, . . . , **pbinom**, . . .)

QUICK REVIEW:

1. Using the built-in functions of R or ggplot, you can -create scatter and line plots, histograms, etc.
   -format the figures with labels, titles, scaling, and with multiple plots
   -create density lines over histogram
   -save them as figures
2. Do the same using ggplot
3. There exist 3 useful types of functions for different distributions that we will use during for this class. The normal distribution versions of the functions are as follows:
   -**rnorm**–random values from distribution
   -**dnorm**–height of probability density function from distribution at a specific point
   -**pnorm**–value of the cumulative density function (CDF) for distribution at a specific point

---

4. Problem

Write a function to simulate 60 different iid samples of 5 coin flips assuming a parameter $p = 0.5$ under a binomial distribution. Within your function calculate the method of moments estimator $T(x) = mean(x)$ for each sample. Use the output of your function to create a histogram of the values taken by the estimator. Save the figure as a png file and upload to CMS (you don't need to upload code).

$$Pr(X = k) = \binom{n}{k} p^k (1-p)^{n-k}$$

for $k = 0, 1, 2, ...n$. The binomial distribution is a discrete probability distribution on the number of k successes in a series of n experiments.

```r
####### PseudoCode #######
library(ggplot2)
binomial_sim <- function(num_of_iid_samples, num_of_observations, prob_of_success) {

  estimator_vector <- vector()

  for (i in 1:num_of_iid_samples) {  # Do something for each sample we want

    # Use rbinom to generate num_of_observations random values sampleed from
    #   a geometric distribution with prob_of_success p
    sample = rbinom(num_of_observations, 1, prob_of_success)
    # Calculate the mean of the sample (maybe using the mean() function? )
    m = mean(sample)
    # Add the mean for that sample to the vector of estimator values
    estimator_vector <- c(estimator_vector, m)

  }

  return(estimator_vector)
}

sim <- binomial_sim(60, 5, 0.5)
sim
# Use any of the methods detailed above to create a histogram
plot_sim <- data.frame(sim)
hist5 <- ggplot(plot_sim, aes(sim)) +
  geom_histogram(aes(y=..density..),
                 bins=20,
                 col='firebrick4',
                 fill='firebrick3') +
  geom_density(colour='black') +
  labs(title='Histogram of Coin Flip Simulation',
       x='Mean Success per simulation',
       y='Frequency')

# Save the histogram as a PNG
setwd('/Users/kateharline/Documents/class_sp19/quant_gen')
ggsave('./problem4_kh694.png', hist5, device='png', dpi = 72)
######## PseudoCode #######
```