Lab Three: Visualizing Data, Probability, the Normal Distribution, and Z Scores

Alex Davis

This lab will cover some of the basics related to visualizing your data, thinking about probability, the normal distribution, and z scores. For this lab, make sure you have the following packages installed:

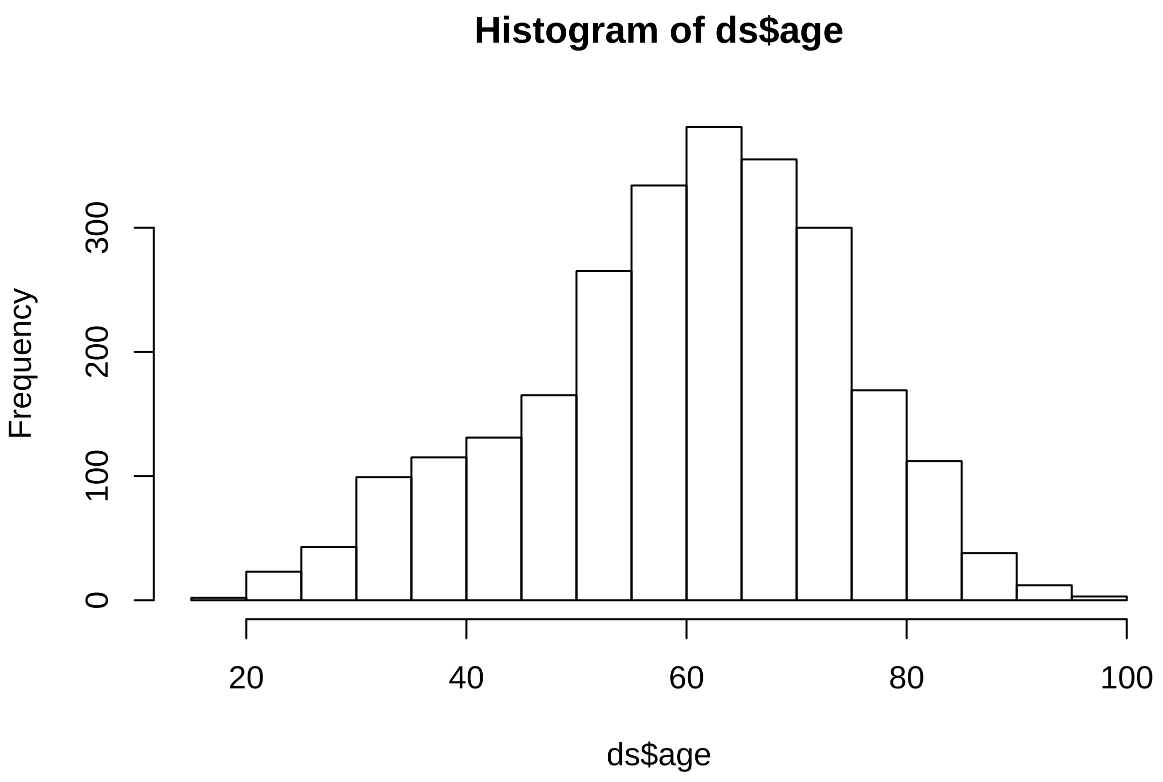
1. sfsmisc
2. psych
3. sm
4. car

Once you have them installed, let's set our working directory, load the data set, and library our packages:

Part One: Histograms and Density

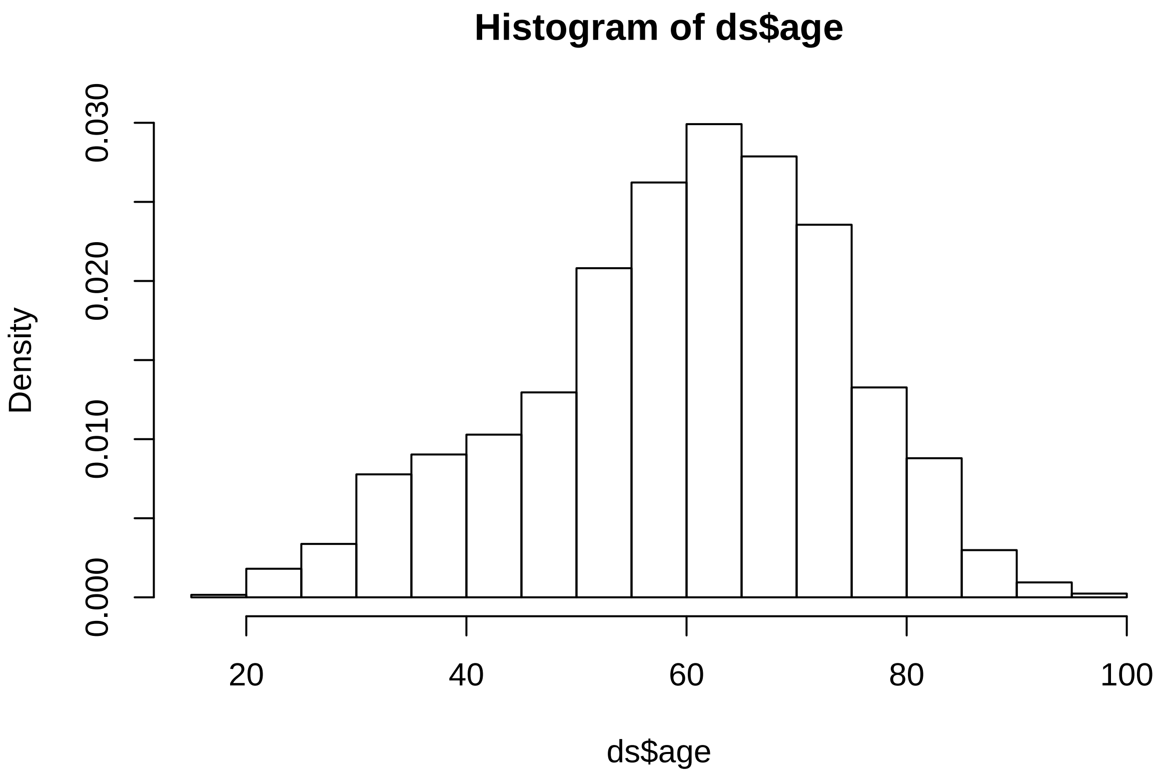
Let's begin by reviewing hisograms. Recall that histograms should be used when visualizing continuous variables. They should not be used when visualizing categorical data. For that you would use a barplot. We can create a histogram in R by using the hist() function, and including which variable you want to examine. Let's look at a continuous variable, age:

```r
hist(ds$age)
```



We just created a basic histogram that displays the frequency of our age variable. However, we can also use a histogram to show density/probability. For this, we use the "probability=TRUE" command inside the hist() function.

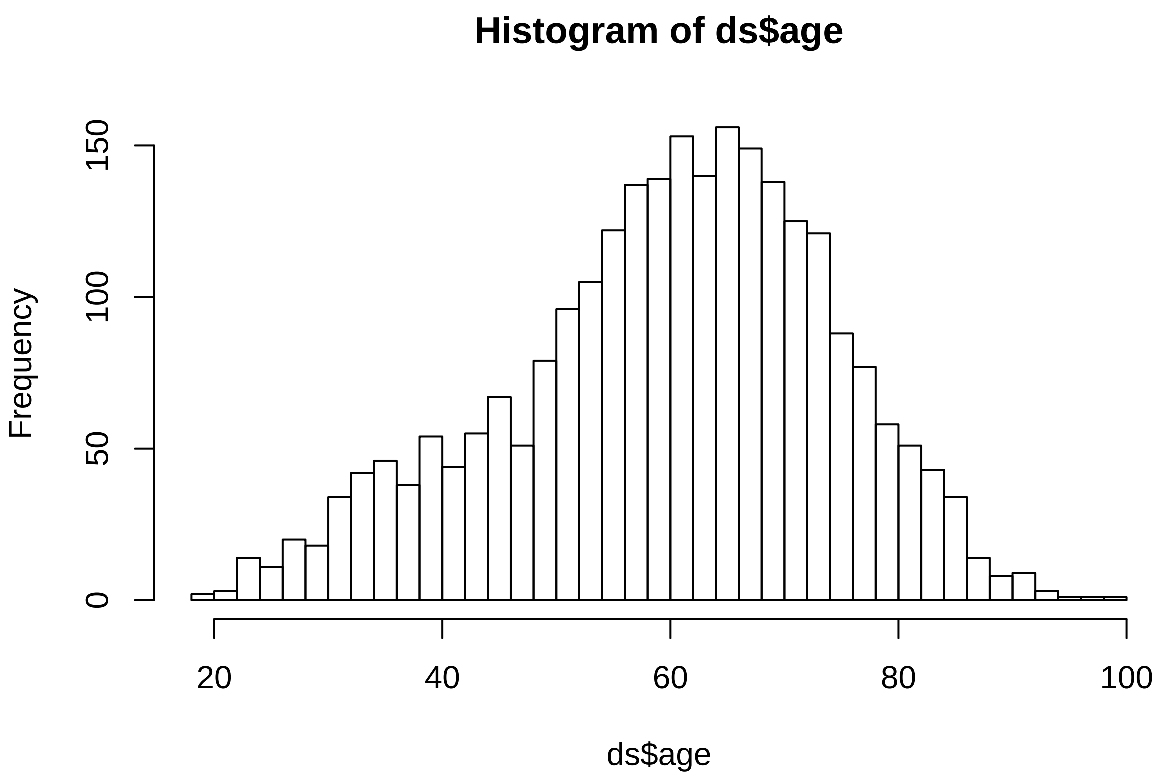
```r
hist(ds$age, probability = TRUE)
```



Notice that this looks like the same visualization as the frequency histogram, but this time the y-axis telling us the density. Look at the highest bar in the histogram, with the age value of about 60-65. The y axis on the frequency histogram tells us that this is the value that occurs the most frequently. The probability distribution tells us that this is also the most likely response from an individual when asked their age.

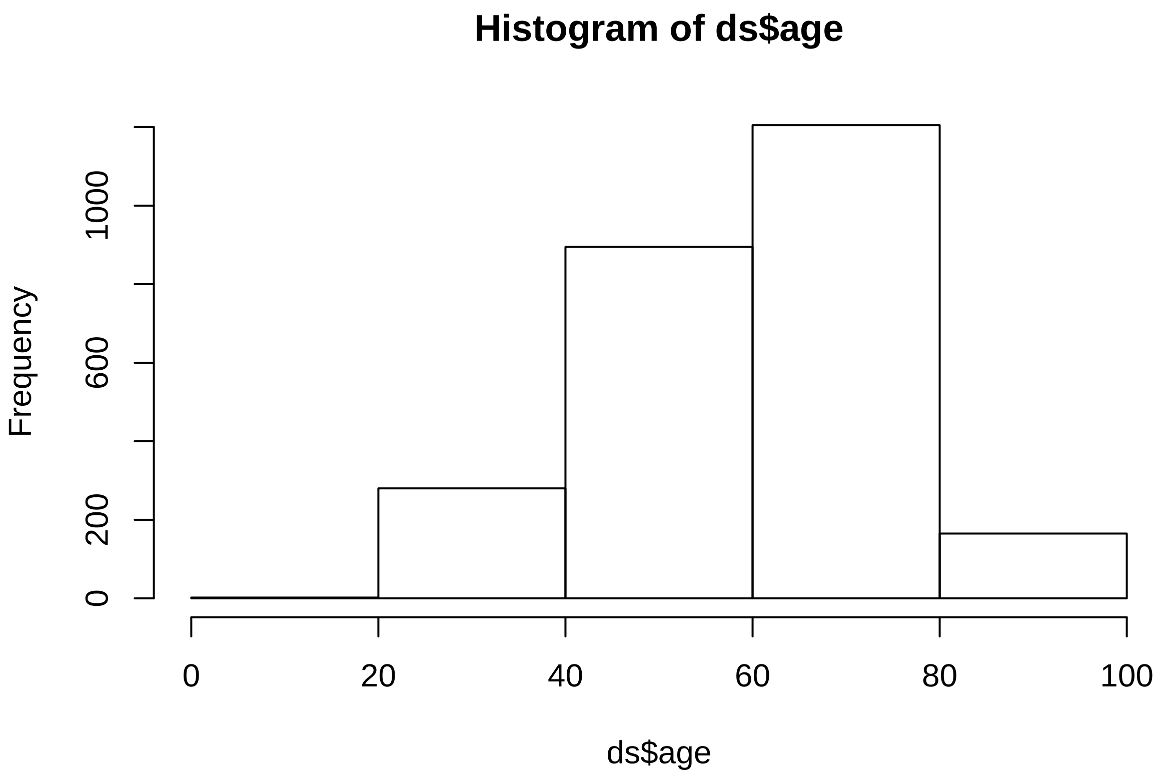
What if you wanted to make the histogram to be more specific about age? In R we can increase the breaks on the x axis. It is relatively simple to do so.

```r
hist(ds$age, breaks = 30)
```



Notice how this histogram has many more bars. Conversely, we can zoom out and make the histogram less specific:

```r
hist(ds$age, breaks = 3)
```

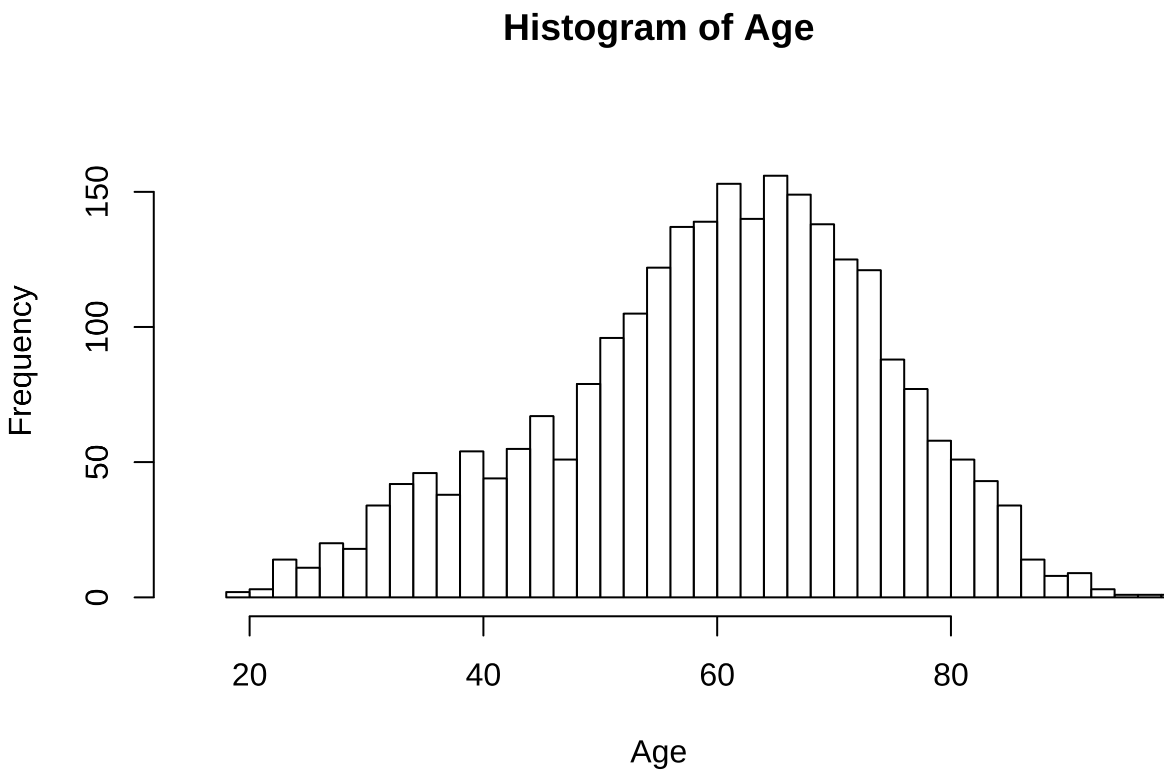


Using various arguments inside the histogram function, we can make the visualization better. These functions can help:

1. xlim =c(min,max) sets the x axis.
2. ylim =c(min,max) sets the y axis.
3. xlab ="X Axis Label" lets you label the x axis.
4. ylab ="Y Axis Label" lets you label the y axis.
5. main ="Title" lets you make a title for the visualization
6. breaks ="# of breaks" lets you specificy how many breaks you want in the histogram.

Let's put all this together and make a good histogram of age:

```
hist(ds$age, breaks=30, xlim=c(15,95), ylim=c(0,175), xlab="Age", ylab = "Frequency",
     main="Histogram of Age")
```

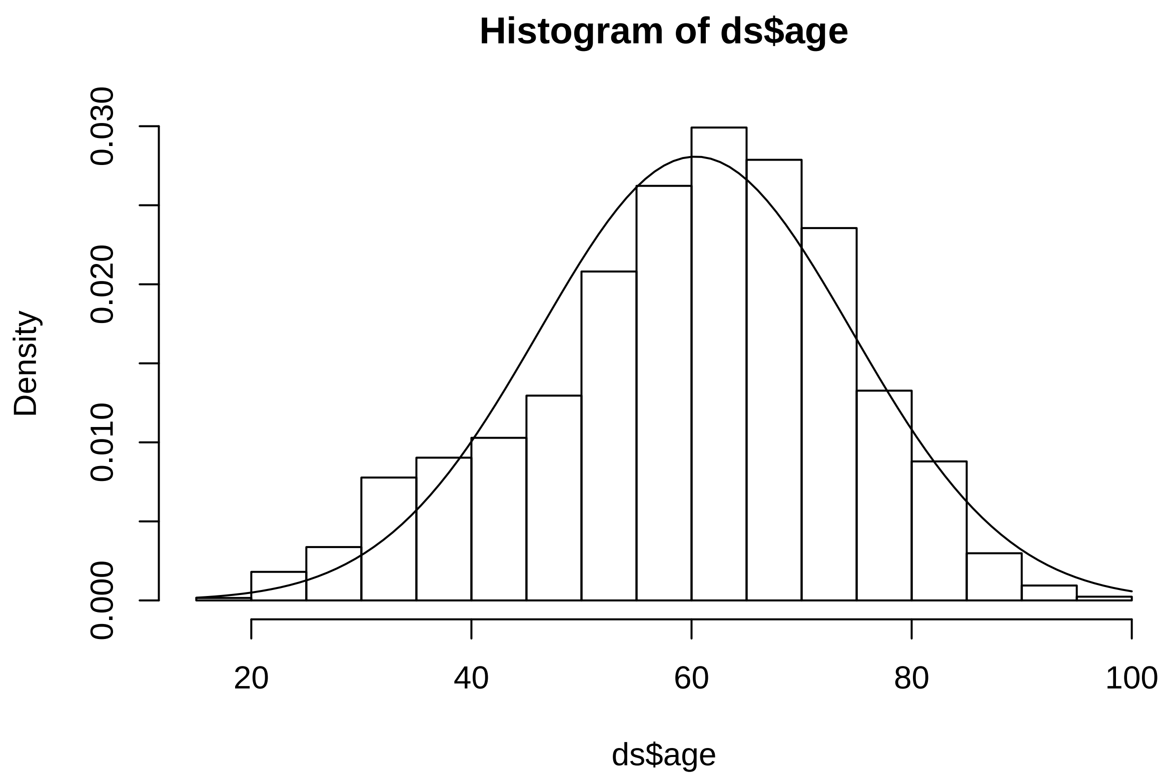


The Normal Distribution and Histograms

Recall that we use the normal distribution when wanting to find probability. In R we can visualize the normal distribution bell curve over a histogram.

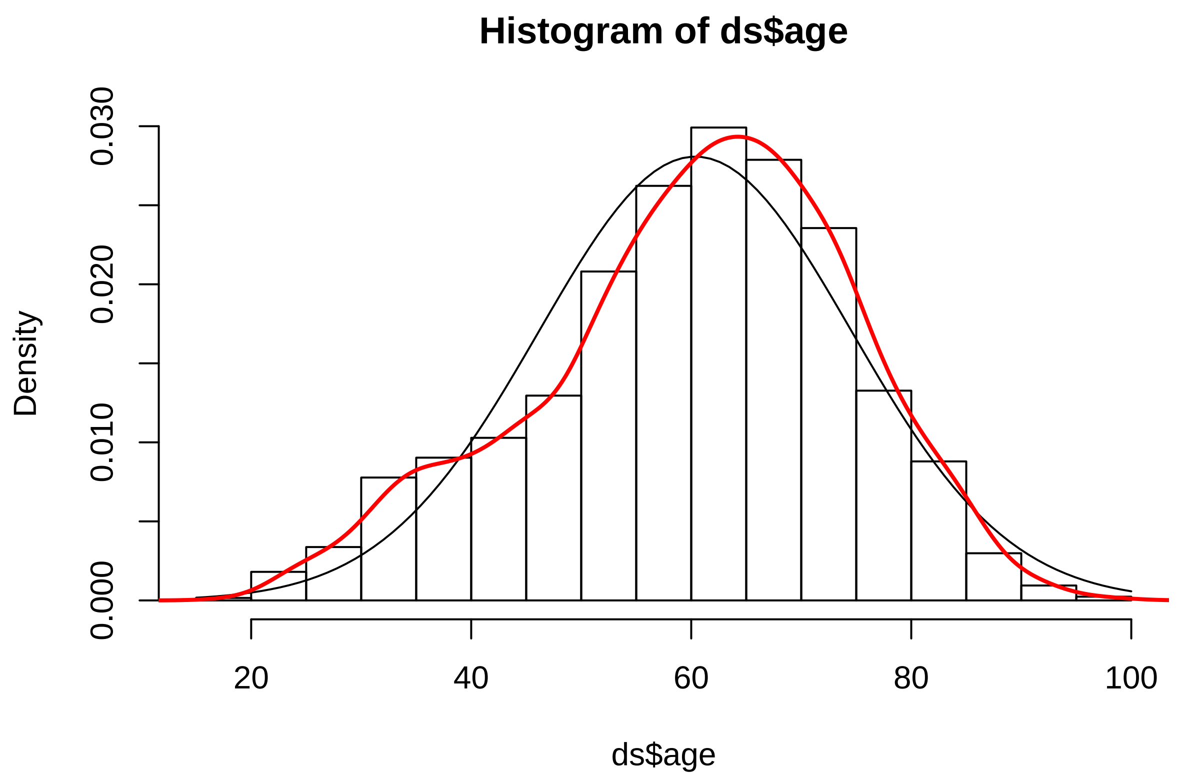
We need to make a density (probability) histogram first. Then we use the curve(), inside which we tell R that we want to make a normal distribution with the mean of our age variable and the standard deviation of our age variable.

```
hist(ds$age, probability = TRUE)
curve(dnorm(x, mean=mean(ds$age, na.rm=T), sd=sd(ds$age, na.rm=T)), add=TRUE)
```



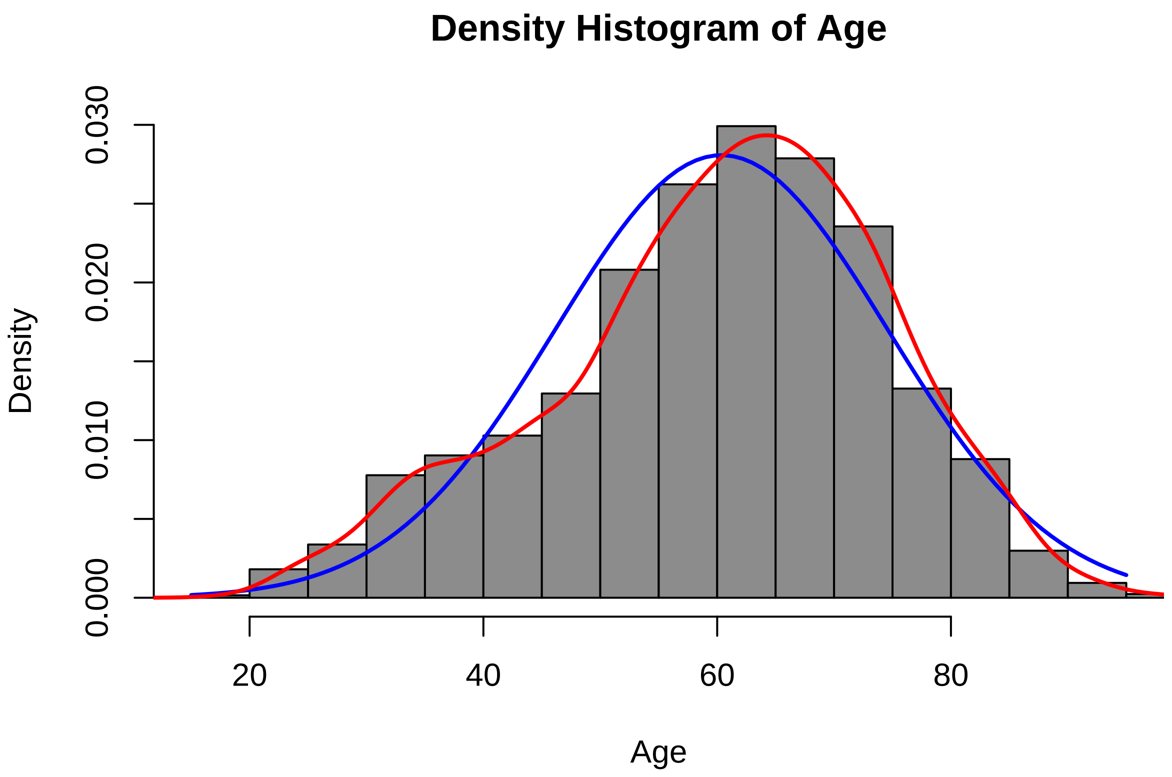
If we are working with non-normal data, we can use the density() smoothing functions to create a line over our histogram. Let's do so, but add it onto the previous visualization:

```r
hist(ds$age, probability = TRUE)
curve(dnorm(x, mean=mean(ds$age, na.rm=T), sd=sd(ds$age, na.rm=T)), add=TRUE)
lines(density(ds$age, na.rm=TRUE), lwd=2, col="red")
```



Let's create one last histogram that combines everything so far, but also includes more color. So we should set an xlim and a ylim, xlab and ylab, main title, the breaks, and we should add a normal curve with the mean and standard deviation of our age varaible and a smoothed line that fits the data a little more. Since we're plotting the normal distribution over our data, we should plot a density histogram instead of a frequency histogram.

```r
hist(ds$age, breaks=20, xlim=c(15,95), ylim=c(0,.03), xlab="Age", ylab = "Density",
     main="Density Histogram of Age", probability = TRUE, col = "gray55")
curve(dnorm(x, mean=mean(ds$age, na.rm=T), sd=sd(ds$age, na.rm=T)), add=TRUE, col="blue", lwd=2)
lines(density(ds$age, na.rm=TRUE), lwd=2, col="red")
```



Part Two: Probability and Distributions

There are many different distributions within R. If you want to see a full list, use the code:

help(Distributions)

When working with the normal distribution, there are four main R functions to use:

- dnorm
- pnorm
- qnorm
- rnorm

The first we'll go over is dnorm. the function dnorm() gives us the height of the probability distribution at a specified point. Use the syntax

dnorm(specified point, mean=**, standard deviation=**)

If you don't specify a mean and standard deviation, R assumes the mean is 0 and the standard deviation is 1. If we wanted to know the height of the probability distribution at point 2 in a distribution with a mean of 4 and and a standard deviation of 1.5, we would find it like this:

```
dnorm(2, mean = 4, sd=1.5)
```

```
## [1] 0.10934
```

Now that we've used dnorm in the abstract, let's try it using our class data set. Let's use the age variable. First we need to know the standard deviation and the mean of the age variable. Luckily we can simply tell R to use the mean and standard deviation using their respective functions. We'll need to include "na.rm=T" inside the functions so that R knows to ignore missing values. Now let's find the height of the pdf at age 65:

```
dnorm(65, mean=mean(ds$age, na.rm = T), sd=sd(ds$age, na.rm=T))
```

```
## [1] 0.02662361
```

Based on our test, the height of the probability distribution at the age 65 point is about .027. This realyl does not provide any intuitive insight, though, because findings the height of the pdf of a random variable does not really tell us anything. If we wanted to find probability, we would need to choose two points of the variable and calculate the probability of someone being in betweenthsoe values. For example, we could find the probability that someone reports being between 65 and 66. To do this we would need to find the integral, or area under the curve, from 65 to 66. We can use the integrate.xy() function to do so:

```r
integrate.xy(density(ds$age)$x,density(ds$age)$y,65,66)
```

```
## [1] 0.02917993
```

We have now found that the probability of someone being between 65 and 66 in our age variable is .029, about a 3% chance.

The second normal probablity function we'll look at is pnorm(). The pnorm() function tells us the probability that we would observe a value less than the one specified. Recall that the cumulative distribution function (cdf) is used for this precise purpose. Similar to dnorm(), in the pnorm() function you specify a value, a mean, and a standard deviation. Again, R defaults to a mean of 0 and a standard deviation of 1 if you do not specify them.

```r
pnorm(5, mean = 6, sd = 2)
```

```
## [1] 0.3085375
```

In a normal distribution with a mean of 6 and a standard deviation of 2, the probability of observing a value less than 5 is about .31. Now let's use pnorm with our class data set. Again let's use age as the variable, but this time let's use pnorm to find the probability that an individual is younger than 65.

```r
pnorm(65, mean=mean(ds$age, na.rm=T), sd=sd(ds$age, na.rm = T))
```

```
## [1] 0.6277983
```

Notice that this probability is much higher, about .63.

Recall that the cdf can be inversed. We use cdf to assess the probability of observing a value less than a specified one, but we can use an inverse cdf to find the probability of observing a value greater than a specified value. In R we use the pnorm() function for this, as well, but we include the argument "lower.tail=FALSE" inside the pnorm() function. Let's find the probability that we'll observe an age greater than 65 in our class data set.

```r
pnorm(65, mean=mean(ds$age, na.rm = T), sd=sd(ds$age, na.rm = T), lower.tail = FALSE)
```

```
## [1] 0.3722017
```

Notice how this probability is lower, about .37, which should make sense.

The next probabiity function we'll learn is qnorm. The qnorm() function is the inverse operation of the pnorm() function. With qnorm(), we specify a probability, on a scale 0 to 1, with given parameters, and R returns the value that should correspond with the probability. Remember, we are inputting the probability that a value is less than, not exact.

```r
qnorm(.3, mean=5, sd=1)
```

```
## [1] 4.475599
```

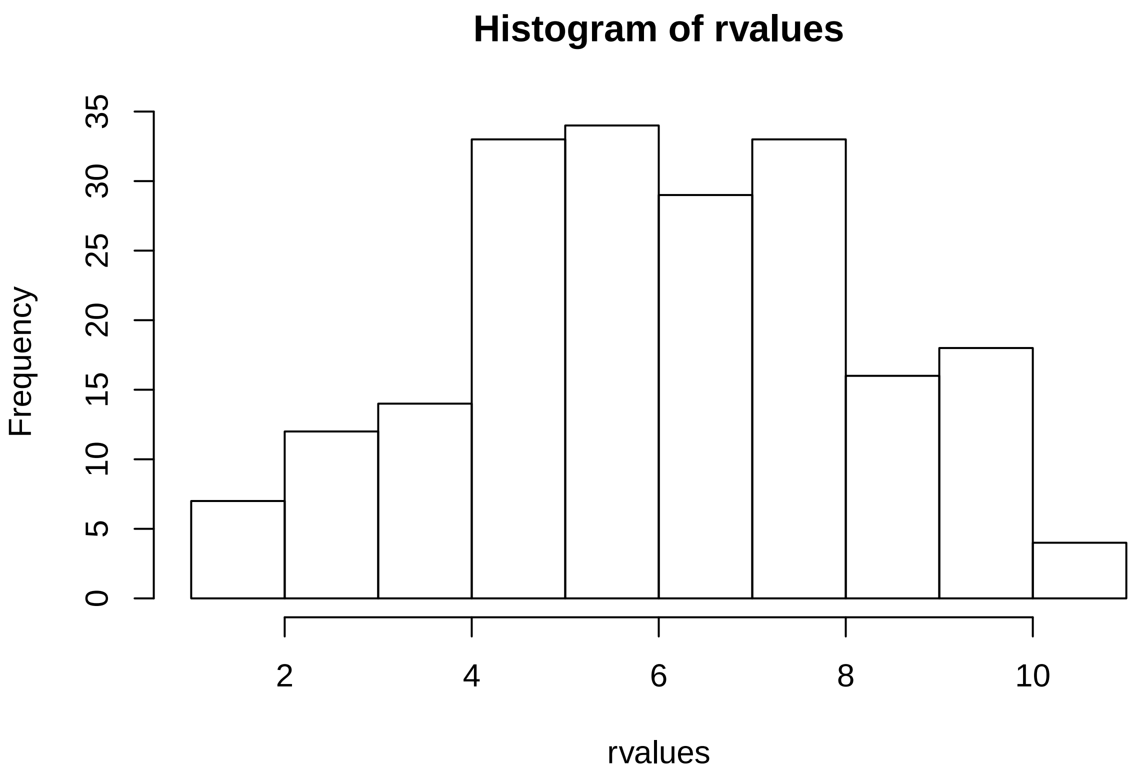
Now let's use the class data set again. Let's find what age has a probability of .4, meaning a .4 probability that we observe values less than it.

```r
qnorm(.4, mean=mean(ds$age, na.rm = T), sd=sd(ds$age, na.rm = T))
```

```
## [1] 56.7677
```

The final probability function we'll look at is the rnorm() function. The rnorm() function generates random values within a given set of parameters. If you don't specify a mean and standard deviation, R defaults to a mean of 0 and a standard deviation of 1. Let's generate some 200 random values with a mean of 6 and a standard deviation of 2, and then use our knowledge of plotting to make a histogram of the values. We'll need to make sure to assign the values to an object.

```r
rvalues <- rnorm(200, mean=6, sd=2)
hist(rvalues)
```



R has similar probability functions for many different distributions. The ones we just learned are for the normal distribtion, but you would also use different functions for the student's t distribution, the binomial distribution, and the chi-squared distribution, to name a few.

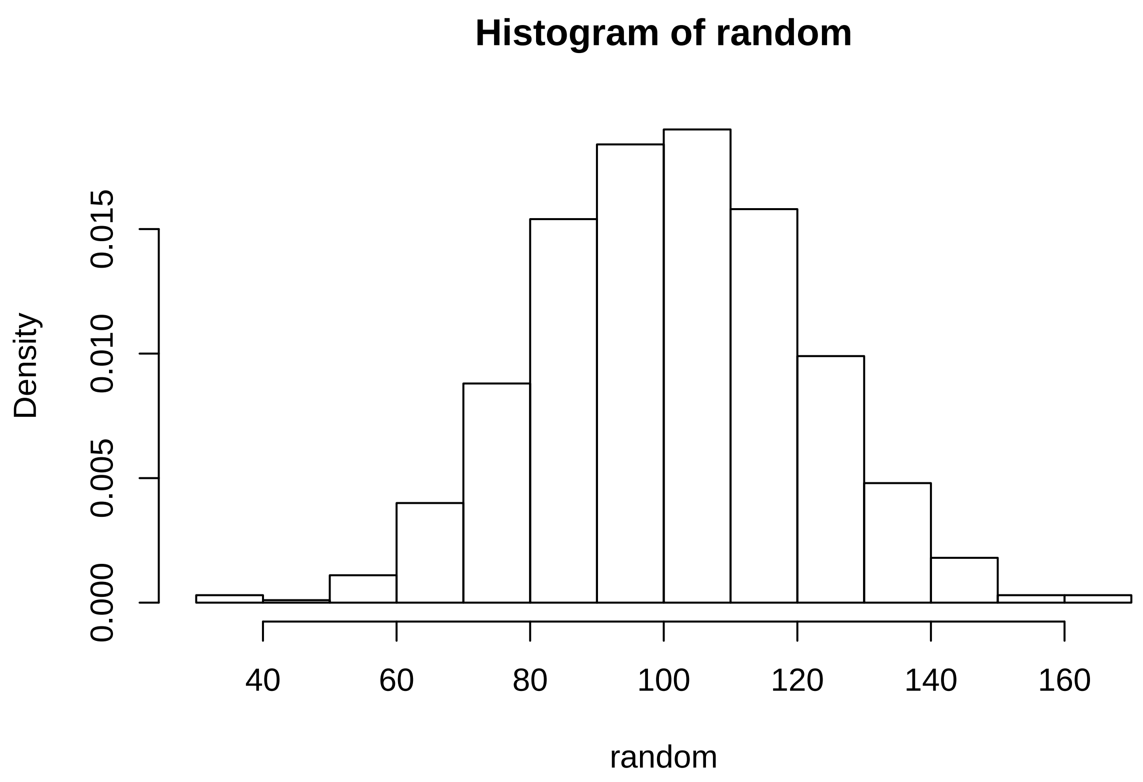
Part Three: Visualizing Normality

Let's move from finding probabilities under the normal distribution to visualizing the normal distribution. First let's use the rnorm() function to generate 1000 random values with a mean of 100 and a standard deviation of 20. We'll assign them to an object called "random".

```r
random <- rnorm(1000, mean = 100, sd = 20)
```

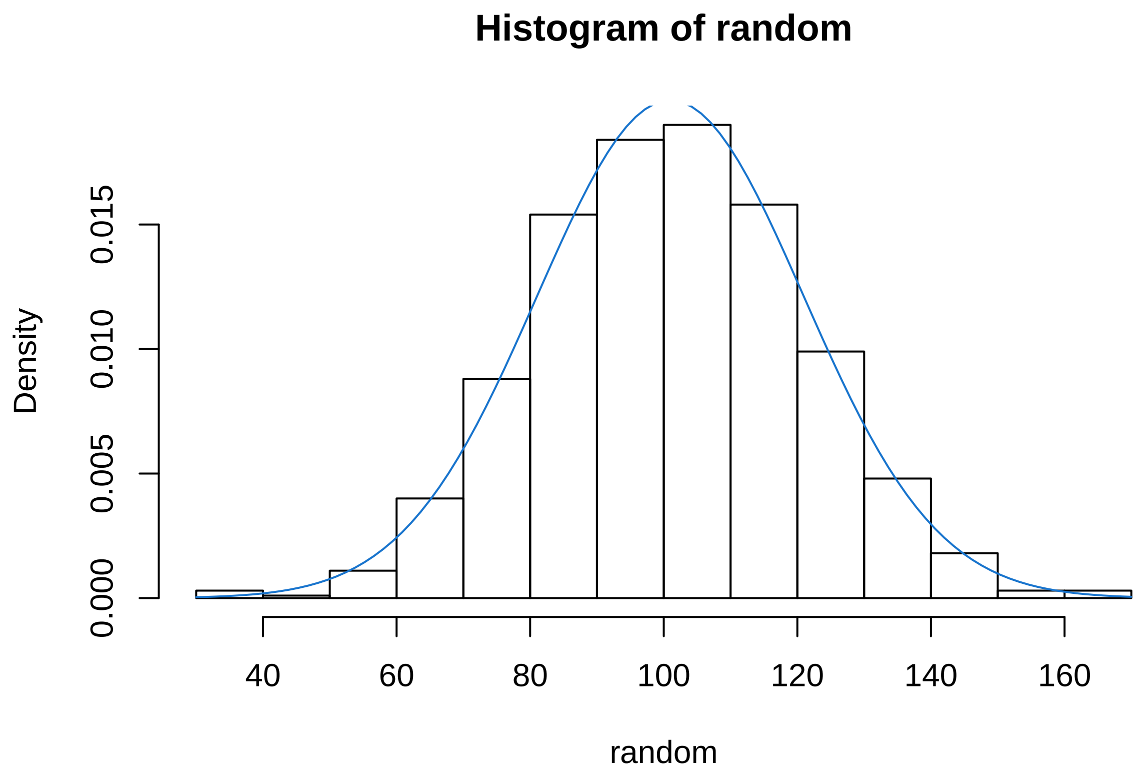
Now let's make a density histogram of the values:

```r
hist(random, probability = TRUE)
```



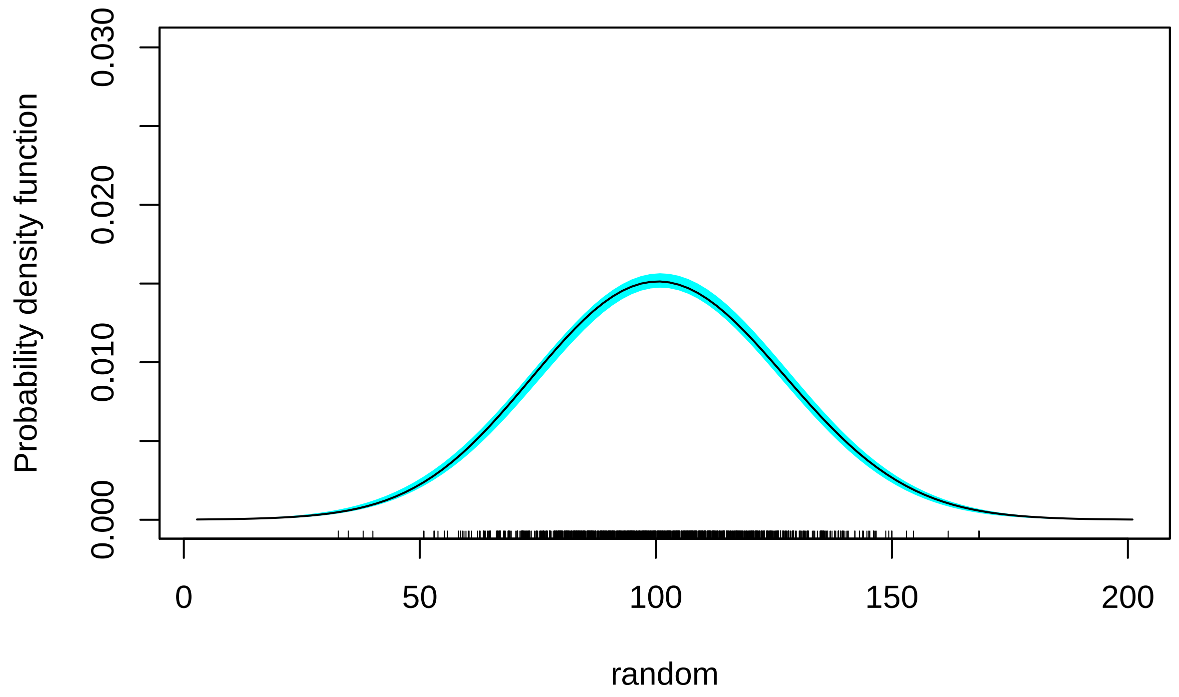
Now let's add the normal bell curve over the histogram, like we did earlier in this lab. We'll need to tell R to use the mean of the random values and the standard deviation of the random values.

```r
hist(random, probability = TRUE)
curve(dnorm(x, mean=mean(random, na.rm=T), sd=sd(random, na.rm=T)), add=TRUE, col="dodgerblue3")
```



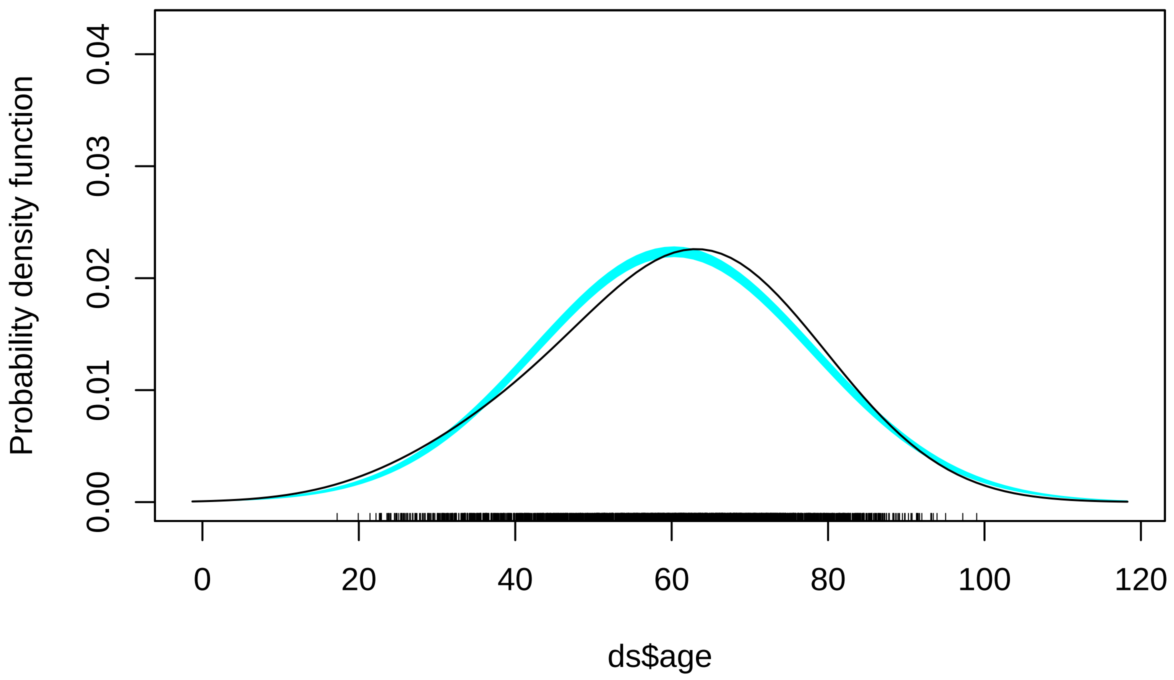
Let's start figuring out how to check if our data is normally distributed. At the beginning of this lab we should have installed and loaded the package "sm". We'll be using the sm.density() function to visualize data and to project what should be the normal distribution of the data, given the mean and standard deviation of the data. Let's start with our randomly generated data. We'll include the model="Normal" argument at the end of the function, so that sm.density knows to use the normal distribution.

```r
sm.density(random, model = "Normal")
```



In the plot above, the black line is the distribution of our randomly genderated data, and the blue line is what the normal distribution of our data should be, given its mean and standard deviation. Since the two overlap, we know that the data is normally distributed. But we already know this, because we told R to generate normally-distributed data. Let's now use a variable from our class data set and check if that data is normally distributed. For the sake of continuity, let's continue using the age variable from our class dataset.

```r
sm.density(ds$age, model = "Normal")
```

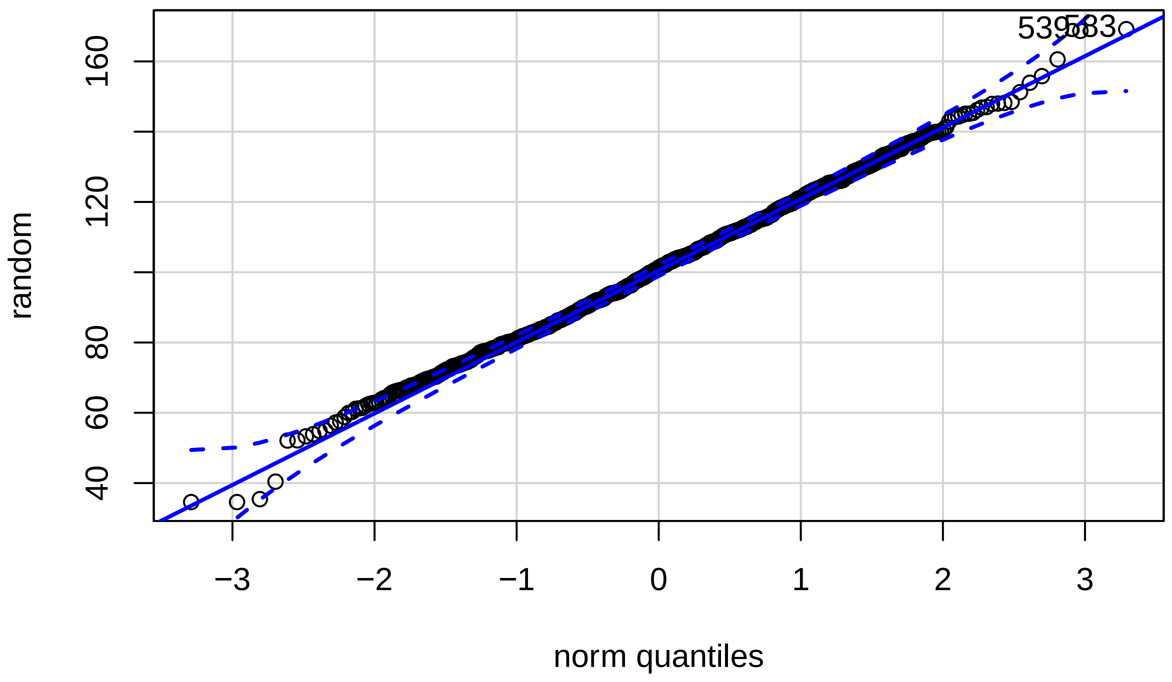


Looking at the black line (the actual distribution of our data) we can see that it is close to being normal, but is still a little skewed. The blue line represents a sort of “confidence band” of normality. It represents what the normal distribution of our data should be given given its mean and standard deviation. Our data distribution overlaps a little with the projected normal distribution at the tails, but is more skewed at the top.

Another way we can check if our data is normally distributed is to make a qqplot. For our purposes, we will use the qqPlot() function from the car package, which should already be installed and loaded. Notice that it is qqPlot, not qqplot.

The qqPlot() functions plots our data based on the quantiles of our variable. If that sounds confusing, the interpretation of the graphic is rather simple. First let’s make the graphic, and then learn how to interpret it. Let’s first use our “random” data, that we generated and is already normally distributed.

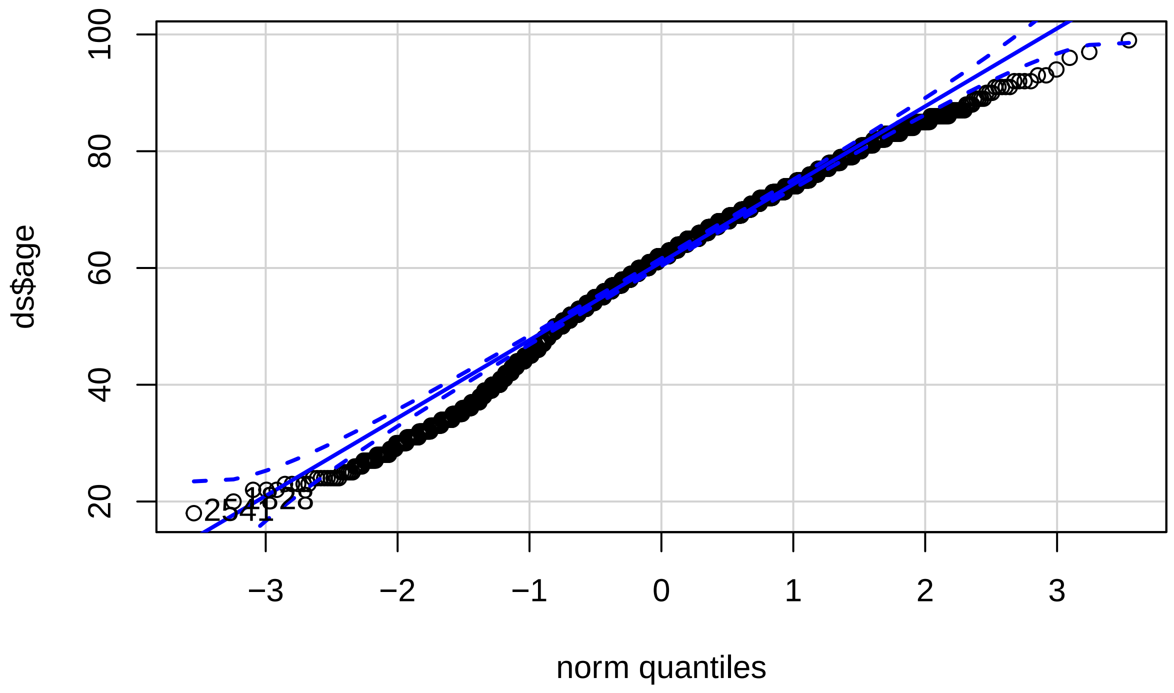
```r
qqPlot(random, distribution = "norm")
```



```
## [1] 583 539
```

In the graphic above, the solid blue line shows where the data should fall if it were normally distributed, and the blue dash lines represent confidence interavls. The individal circles show where our data fall. If it falls within the intervals, our data might be normally-distributed. It looks like our data is normal, as it should be, since we generated it to be. Again, let's now use our class data set "age" variable and check its normality.

```
qqPlot(ds$age, distribution = "norm")
```

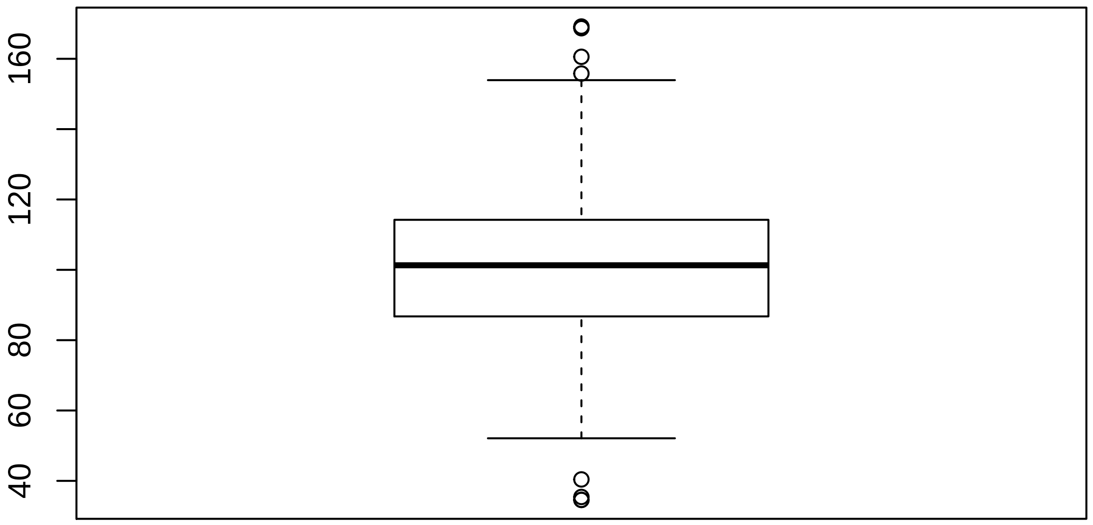


```
## [1] 2541 1828
```

This visualization shows what we already suspected from the sm.density() plot: the age variable is close to normally-distributed, and in some places falls within the bounds, but ventures from it at other points.

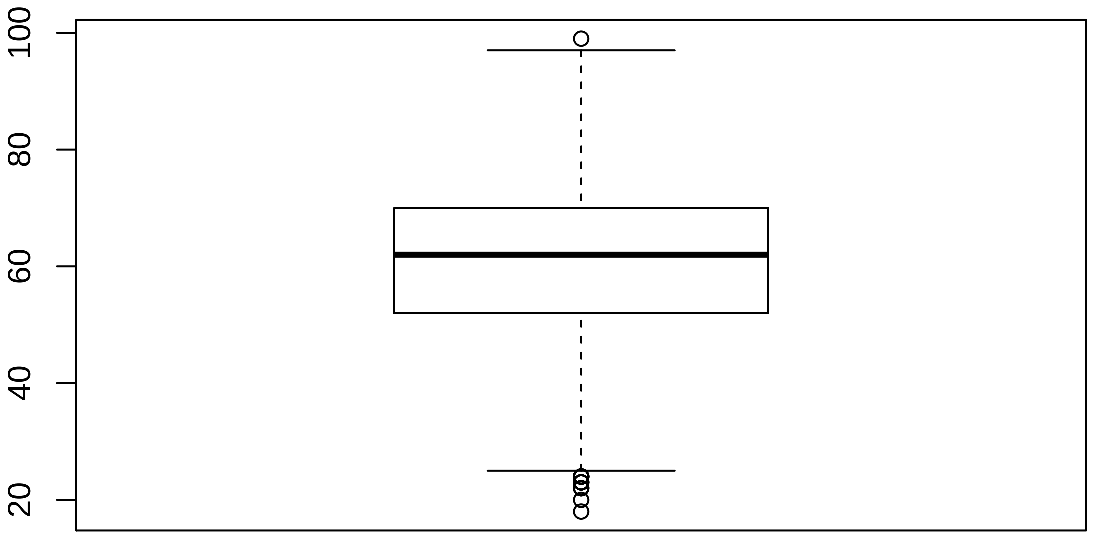
The last method we'll look at for exploring normality is the boxplot. The boxplot plots the quartiles and the median, and shows us individual unique values at the edges of our data. Let's look at a boxplot of our random data:

```
boxplot(random)
```



When looking at a box plot, the middle line represents the middle quartile, or the median. The area between the median line and the line below it represents the second quartile. The area from the bottom of the second quartile line to the very bottom line, or "whisker" is the first quartile. Above the median, the area between the median line and the next line above it is the third quartile, and the area above that, and below the top whisker, is the fourth quartile. Notice that for the boxplot of our randomly generated data, the quartiles are relatively balanced. Let's now make a boxplot for our age variable:

```
boxplot(ds$age)
```



Notice how with this boxplot, it is relatively balanced, but not as much as our purely normally-distributed data in the previous visualization.

Part Four: Z Scores

Recall that the standard deviation can be thought of as the average distance to the mean of a set of data, it measures the level of deviation in the data. We can use a z score to measure how many standard deviations away from the mean a particular point of data is. Let's keep using the age variable from our class data set and calculate z scores.

First we need to create a new variable in our data set that scales the age variable by standard deviations. Luckily this is very easy to do in R. Let's name the new variable z.age:

```
ds$z.age <- scale(ds$age)
```

Now we need to pull specific observations of data out and then we can see its z score. We will use a particular method of subsetting data, in which we tell R specific parameters based on the column and row names. . Follow the basic syntax used in the example below to do this for other specific parameters. Let's find the z score of a woman who is younger than 19 years old.

```
ds[which(ds$f.gender=='Women' & ds$age < 19), c("f.gender", "age", "z.age")]
```

```
##      f.gender age    z.age
## 2541    Women  18 -2.981749
```

We find that there, in our class data, there is one woman, age 18, and her z score for age is -2.98. This means that her age is 2.98 standard deviations below the mean. Let's do a shorthand calculation to find just many years below the mean the z score suggests.

First lets find the standard deviation of our age variable:

```
sd(ds$age, na.rm = TRUE)
```

```
## [1] 14.20894
```

The standard deviation is about 14.21 years. Now let's multiple the absolute value of the z score we found above by the standard deviation of the age variable:

```
2.981748*sd(ds$age, na.rm = TRUE)
```

```
## [1] 42.36748
```

This tells us that the individual we observed is about 43.38 years below the average age in our data set.

Following the syntax used in the first z score example, we can find the z score based on any given parameters. Z scores are useful because they give us a scaled way of comparing the deviance in variables.