The Normal Curve

Stat 131A, Fall 2018, Prof. Sanchez

Learning Objectives

- Becoming familiar with the normal curve
- Intro to the functions `dnorm()`, `pnorm()`, and `qnorm()`
- How to find areas under the normal curve using R
- Converting values to standard units

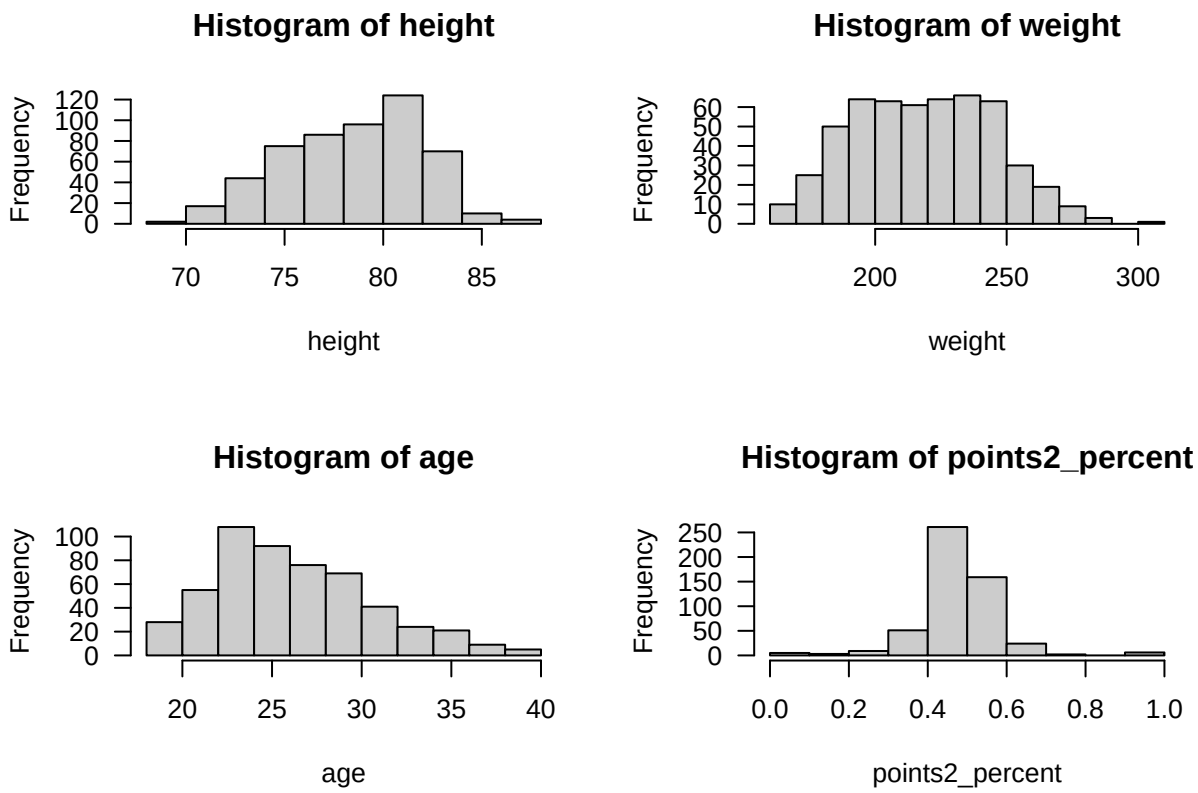
Introduction

Let's look at the distributions of some variables in the data of NBA players:

```
# assembling the URL of the CSV file
repo = 'https://raw.githubusercontent.com/ucb-introstat/introstat-fall-2018/'
datafile = 'master/data/nba_players.csv'
url = paste0(repo, datafile)

# read in data set
nba = read.csv(url)
```

More specifically, let's take a peek at the histograms of variables `height`, `weight`, `age`, `points2_percent`



- `height` seems to have a slightly left skewed distribution.
- `weight` looks roughly symmetric.
- `age` has a right skewed distribution.
- `points2_percent` appears to be fairly symmetric.

These distributions are examples of some of the possible patterns that you will find when describing data in real life. If you are lucky, you may even get to see a perfect symmetric distribution one day.

Among the wide range of distribution shapes that we encounter when looking at data, one special pattern has received most of the attention: the so-called *symmetric bell-shaped* or mound-shaped distribution, like that of `points2_percent` and `weight`. It is true that these two histograms are far from perfect symmetry, but we can put them within the *fairly* bell-shaped category.

Normal Curve

It turns out that there is one mathematical function that fits (density) histograms having a symmetric bell-shaped pattern: the famous **normal curve** given by the following equation

$$y = \frac{1}{\sqrt{2\pi}} e^{-\frac{1}{2}x^2}$$

This equation, also known as the Laplace-Gaussian curve, was first discovered by Abraham

de Moivre (circa 1720) while working on the first problems about probability. However, his work around the normal equation went unnoticed for many years. By the time historians realized he had been the first person to come up with the normal equation, most people had attributed authorship to either French scholar Pierre-Simon Laplace and/or German mathematician Carl Friedrich Gauss.

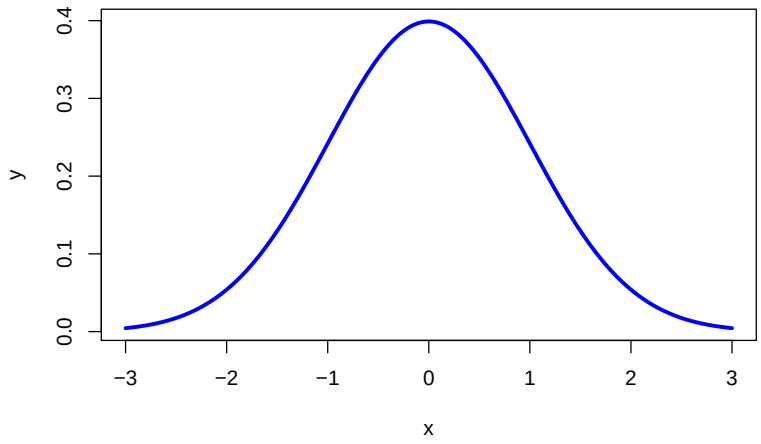
In the past, before the 1880s, the curve was referred to as the *Error curve*, because of its application around the errors from measurements in astronomy. The name *normal* appeared around the late 1870s and early 1880s, where British biometricians like Francis Galton, and later on his disciple Karl Pearson, together with Ronald Fisher, popularized the word *normal*. Galton never explained why he used the term "normal" although it seems that he was implying the sense of conforming to a norm (i.e. a standard, model, pattern, type).

Plotting the Normal Curve in R

You can use R to obtain a graph of the normal curve. One approach is to generate values for the x-axis, and then use the equation of the normal curve to obtain values for the y-axis:

```
x = seq(from = -3, to = 3, by = 0.01)
y = (1/sqrt(2 * pi)) * exp(-(x^2)/2)

plot(x, y, type = "l", lwd = 3, col = "blue")
```



First we generate a vector `x` with some values for the x-axis ranging from -3 to 3. Then we use `x` to find the heights of the `y` variable. Finally, we use the values in `x` and `y` as coordinates of the `plot()`. The argument `type = 'l'` is used to graph a line instead of dots. The argument `lwd` allows you to define the width of the line. And `col` lets you define a color.

Normal Distribution Functions

Instead of working with the equation `y = (1/sqrt(2 * pi)) * exp(-(x^2)/2)`, R has a family of four functions dedicated to the normal curve:

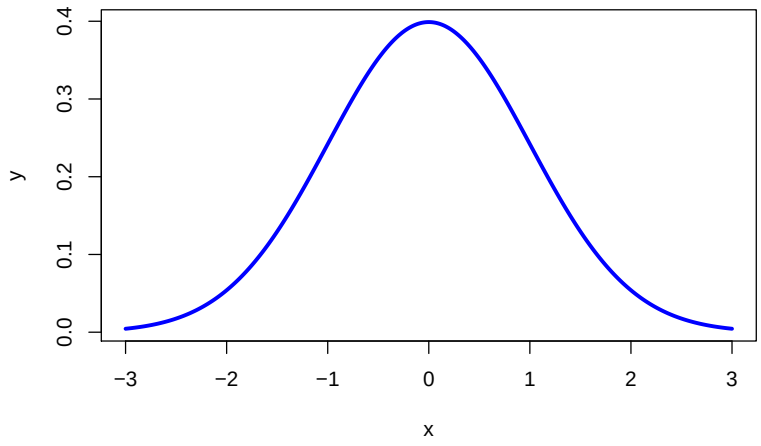
- `dnorm()` density function
- `pnorm()` distribution function
- `qnorm()` quantile function
- `rnorm()` random number generator function

Heights of the curve with `dnorm()`

The function `dnorm()` is the **density** function. This is actually the function that lets you find the height of the curve (i.e. y values). Instead of manually coding the normal equation, you can use `dnorm()` and get the previously obtained graph like this:

```
x = seq(from = -3, to = 3, by = 0.01)
y = dnorm(x)

plot(x, y, type = "l", lwd = 3, col = "blue")
```



Areas under the curve with `pnorm()`

The function `pnorm()` is the distribution function. By default, `pnorm()` returns the area under the curve to the **left** of a specified `x` value. For instance, the area to the left of 0 is 0.5 or 50%:

```
pnorm(0)
```

```
## [1] 0.5
```

Try `pnorm()` with these values

```
pnorm(-2)
pnorm(-1)
pnorm(1)
pnorm(2)
```

You can also use `pnorm()` to find areas under the normal curve to the **right** of a specific `x` value. This is done by using the argument `lower.tail = FALSE`:

```
# area to the right of 1
pnorm(1, lower.tail = FALSE)
```

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

Try finding the areas to the right of:

```
pnorm(-2.5, lower.tail = FALSE)
pnorm(-2, lower.tail = FALSE)
pnorm(0.5, lower.tail = FALSE)
pnorm(1.5, lower.tail = FALSE)
```

Sometimes you need to find areas in between two z values. For instance, the area between -1 and 1 (which is about 68%). Finding this type of areas involves subtracting the larger area to the left of 1 minus the smaller area to the left of -1:

```
# area between -1 and 1
pnorm(1) - pnorm(-1)
```

```
## [1] 0.6826895
```

What abot the area between -2 and 2?

```
# area between -2 and 2
pnorm(2) - pnorm(-2)
```

```
## [1] 0.9544997
```

Z values of a given area with `qnorm()`

The function `qnorm()` is the quantile function. You can think of this function as the inverse of `pnorm()`. That is, for a given area under the curve, use `qnorm()` to find what is the corresponding `z` value (i.e. value on the x-axis):

```
# z-value such that the area to its left is 0.5
qnorm(0.5)
```

```
## [1] 0
```

```
# z-value such that the area to its left is 0.3
qnorm(0.3)
```

```
## [1] -0.5244005
```

Likewise, you can use the argument `lower.tail = FALSE` to find values given a right-tail area:

```r
# z-value such that the area to its right is 0.5
qnorm(0.5, lower.tail = FALSE)
```

```
## [1] 0
```

```r
# z-value such that the area to its right is 0.3
qnorm(0.3, lower.tail = FALSE)
```

```
## [1] 0.5244005
```

Standard Units

In real life, most variables will be measured in some scale: `height` measured in inches, `weight` measured in ounces, `age` measured in years, `points2_percent` measured in percentage. To be able to use the normal curve as an approximation for symmetric bell-shaped distributions, you will need to convert the original units into **standard units** (SU).

Recall that the conversion formula from x to standard units is:

$$SU = \frac{x - avg}{SD}$$

Let's see how you could convert `weight` values to SU using R. First we need to obtain find the average and standard deviation of `weight`:

```r
# average weight
avg_weight = mean(nba$weight)
avg_weight
```

```
## [1] 220.5852
```

```r
# SD weight
# (remember to use correction factor)
n = nrow(nba)
sd_weight = sqrt((n-1)/n) * sd(nba$weight)
sd_weight
```

```
## [1] 26.81206
```

To convert the weights of the players to standard units, subtract the average and then divide by the SD:

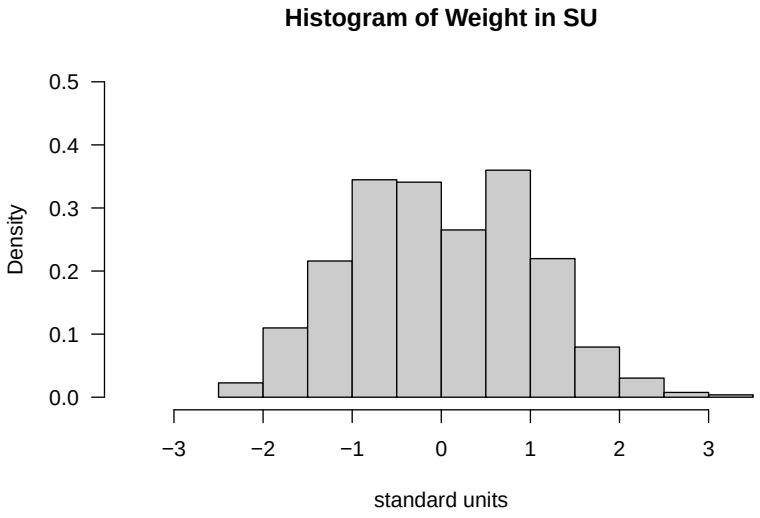
```r
# weight in SU
su_weight = (nba$weight - avg_weight) / sd_weight

# weights in SU of first 5 players
su_weight[1:5]
```

```
## [1]  0.9105891 -1.8120660 -1.2899130 -1.3272096 -0.7304633
```

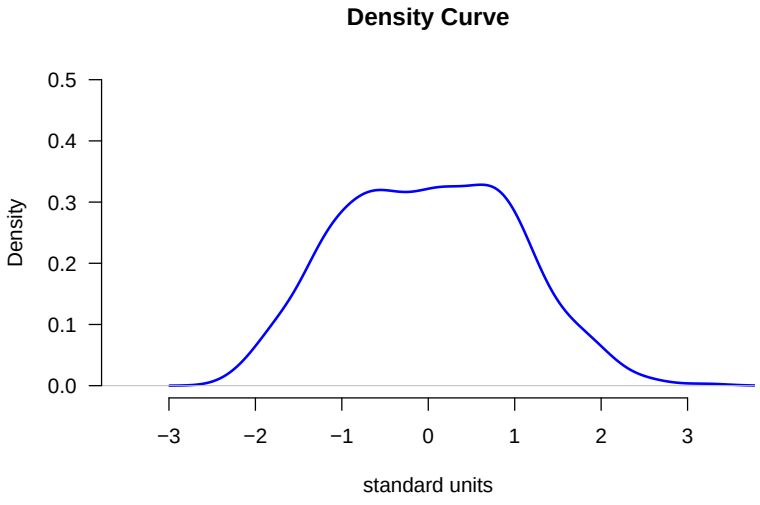
How does the histogram for `su_weight` look like?

```
# density histogram
hist(su_weight, las = 1, col = 'gray80', probability = TRUE,
     ylim = c(0, 0.5), xlim = c(-3.5, 3.5),
     main = 'Histogram of Weight in SU', xlab = 'standard units')
```



An alternative picture of the distribution of `su_weight` can be obtained by plotting a kernel density curve:

```
dens_weight = density(su_weight)
plot(dens_weight, axes = FALSE, ylim = c(0, 0.5), xlim = c(-3.5, 3.5),
     main = 'Density Curve', xlab = 'standard units', lwd = 2, col = 'blue')
# x-axis
axis(side = 1)
# y-axis
axis(side = 2, las = 1)
```



Looking at both the histogram, and the kernel density curve, the shape of the distribution is symmertric but it does not have a peak around zero. You can say that it has moero of a plateau or flat peak.

Using Normal Approximation

Although `weight` does not have the central peak, we can try to see how good the normal curve approximates its distribution. From the attributes of the normal curve, we know that 50% of players should have a height below `avg_weight`. We can directly check what is the proportion of players below `avg_weight`:

```
# proportion of players below average weight
sum(nba$weight <= avg_weight) / n
```

```
## [1] 0.5170455
```

This confirms that `weight` (and `su_weight`) does have a symmetric shape.

From the empirical 68-95-99.7 rule, we know that about 68% of players should have weights between 220.59 plus-minus 26.81, that is, between 193.77 and 247.4

```
weight_minus = avg_weight - sd_weight
weight_plus = avg_weight + sd_weight

# proportion of players within 1 SD from average weight
sum(nba$weight <= weight_plus & nba$weight >= weight_minus) / n
```

```
## [1] 0.655303
```

As you can tell, the proportion of players around 1 SD is not 68% but 65%. However, the difference of 3% is not that big.

Asumming Normality ...

Let's pretend that `wheight` does have a symmetric bell-shaped distribution, and that we are interested in finding the proportion of players with weights below 200 pounds.

You can use `pnorm()` to find such proportion, and without having to convert to standard units. All you need to do is specify the `mean` and `sd` arguments with the corresponding average and SD values, respectively:

```
# proportion of players with weight below 200 pounds
pnorm(200, mean = avg_weight, sd = sd_weight)
```

```
## [1] 0.2213149
```

To find the proportion of players with weights above 230 pounds, include the argument `lower.tail = FALSE`:

```r
# proportion of players with weight above 230 pounds
pnorm(230, mean = avg_weight, sd = sd_weight, lower.tail = FALSE)
```

```
## [1] 0.3627419
```

You can also use `qnorm()` to find what would be the corresponding weight such that 60% of players are below it:

```r
qnorm(0.6, mean = avg_weight, sd = sd_weight)
```

```
## [1] 227.378
```

Or what is the weight wuch that 35% of players are above it:

```r
qnorm(0.35, mean = avg_weight, sd = sd_weight, lower.tail = FALSE)
```

```
## [1] 230.9165
```