The Normal Curve

Intro to Stats, Spring 2017

Prof. Gaston Sanchez

Learning Objectives

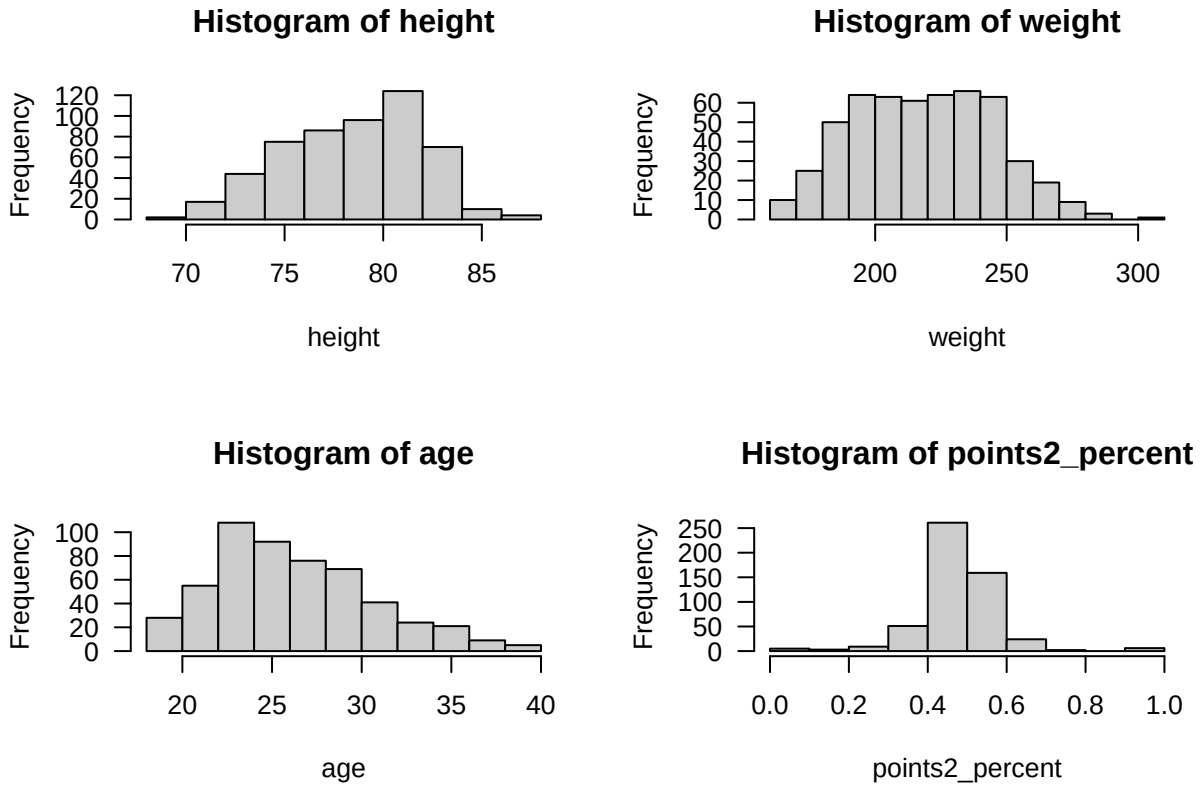
- Becoming familiar with the normal curve
- Intro to the functions `dnorm()`, `pnorm()`, and `qnorm()`
- How to find areas under the normal curve using R
- Converting values to standard units

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-spring-2017/'
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, and `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 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.

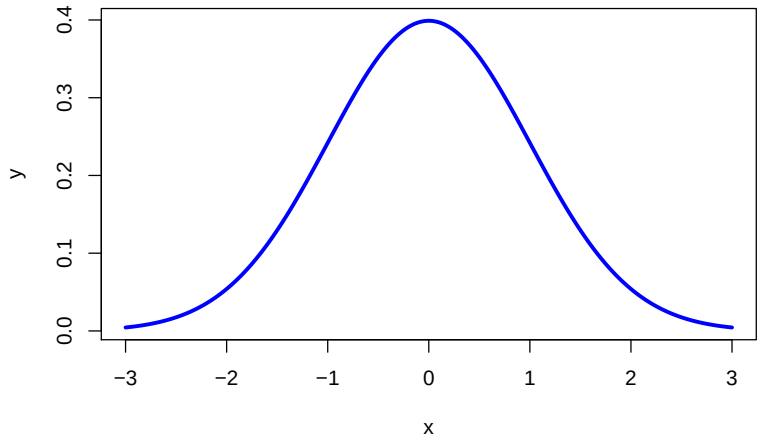
In the past, before the 1880s, the curve was referred to as the *Error curve*, because of its application around the errors from measurements in astronomy. The name *normal* appeared around the late 1870s and early 1880s, where British biometricians like Francis Galton, and later on his disciple Karl 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 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 to define the width of the line.

Normal Distribution Functions

R provides functions for different (probability) distributions. In the case of the Normal distribution, there is a family of four functions:

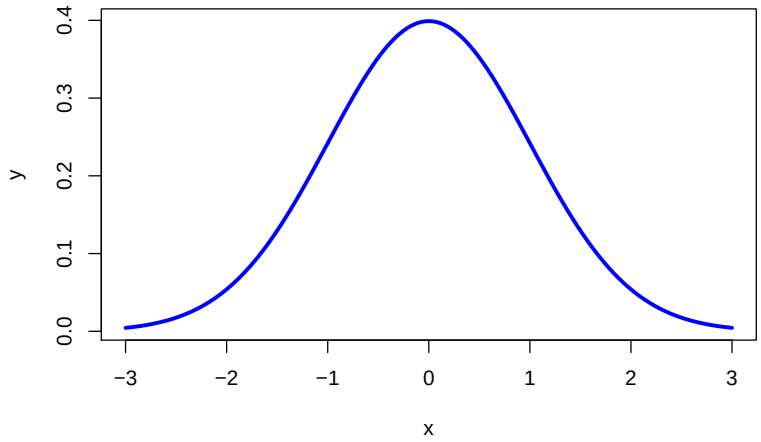
- `dnorm()` density function
- `pnorm()` distribution function
- `qnorm()` quantile function
- `rnorm()` random number generator function

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:

```
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 ‘ 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):

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

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

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

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

```r
# x-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).

Average and standard deviation of `height`

```r
# average height
avg_height = mean(nba$height)
avg_height
```

```
## [1] 78.9678
```

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

```
## [1] 3.484395
```

To convert the heights of the first ten players to standard units, we need to subtract the average and divide by the SD:

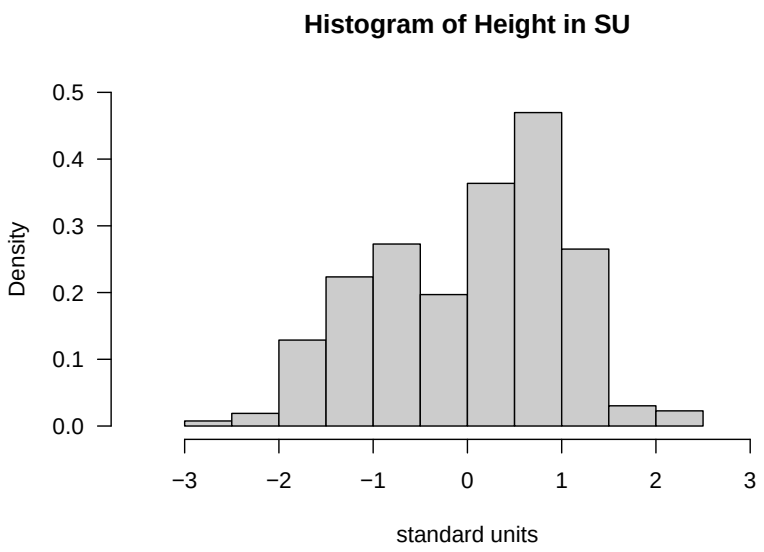
```r
su_height = (nba$height - avg_height) / sd_height

# heights in SU of first 5 players
su_height[1:5]
```

```
## [1]  0.8702219 -1.7127229 -1.4257290 -0.2777535 -0.5647474
```

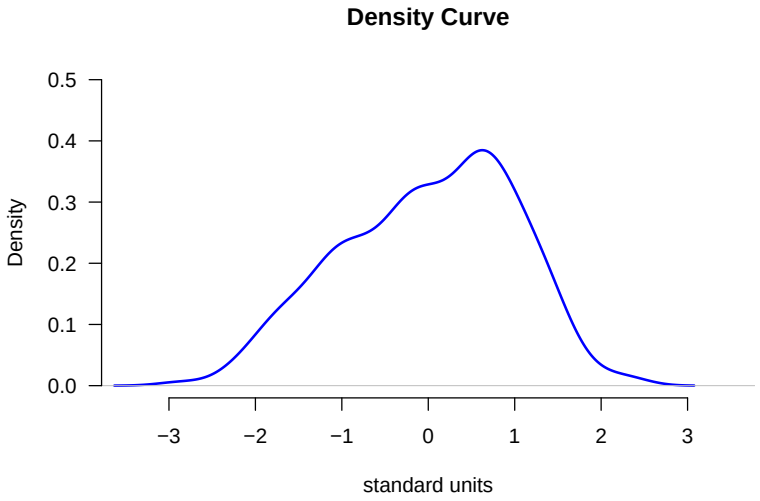
Histogram

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



Curve

```r
dens_height = density(su_height)
plot(dens_height, 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)
```



Using Normal Approximation

About 50% of players should have a height below `avg_height`

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

```
## [1] 0.4242424
```

About 68% of players should have heights between 78.967803 plus-minus 3.48, that is between 75.48 and 82.45

```
height_minus = avg_height - sd_height
height_plus = avg_height + sd_height

# proportion of players within 1 SD from average height
sum(nba$height <= height_plus & nba$height >= height_minus) / n
```

```
## [1] 0.6515152
```