Measures of Centrality and Spread

JuJu

For this example we will use the ***htwt*** data. In particular, we will look at the weights (***Weight***) of the 20 individuals in the data set.

 htwt = read.csv("http://facweb1.redlands.edu/fac/jim\_bentley/downloads/math111/htwt.csv")
 head(htwt)

## Height Weight Group
## 1 64 159 1
## 2 63 155 2
## 3 67 157 2
## 4 60 125 1
## 5 52 103 2
## 6 58 122 2

## Measures of Centrality

The raw weight data for the sample data set can be seen by asking for the proper column. Three equivalent ways of accessing the second, ***Weight*** column are given below.

 htwt$Weight

## [1] 159 155 157 125 103 122 101 82 228 199 195 110 191 151 119 119 112 87 190
## [20] 87

 htwt[,"Weight"]

## [1] 159 155 157 125 103 122 101 82 228 199 195 110 191 151 119 119 112 87 190
## [20] 87

 htwt[,2]

## [1] 159 155 157 125 103 122 101 82 228 199 195 110 191 151 119 119 112 87 190
## [20] 87

### The Mean or Average

The mean is a middle value that represents middle by being the balance point of the data. It is computed by summing all of the values and then dividing by the number of values.

R provides the function ***mean*** to compute the average.

 # First we add all of the values
 sumx = sum(htwt$Weight)
 sumx

## [1] 2792

 # Now divide by the number of observations
 n = length(htwt$Weight)
 n

## [1] 20

 Wt.mean = sumx / n
 Wt.mean

## [1] 139.6

 # Or we can use the internal function mean
 mean(htwt$Weight)

## [1] 139.6

### The Median

The median is **a** value that divides the sorted data in half. If there are an even number of observations, the median is not unique. Convention dictates that in the case of an even number of observations we take the average of the two middle values.

If there are an odd number of observations, we use the (n+1)/2 observation in the sorted data. If the number of observations is even, we average the (n/2) and (n/2)+1 observations. Since the ***Weight*** data has 20 observations, we will use the second method.

R has the built-in function ***median*** that also provides the median.

 # For those who like to program...
 # Create a function to figure out if an integer i is odd
 odd = function(i){
 # If n is not an integer then stop
 if (!is.integer(i)){stop("Not an integer.")}
 # Otherwise, if the division by 2 has no remainder, then i is even
 else if (i %% 2 == 0) {return(FALSE)} else {return(TRUE)}
 }
 # Get the number of observations and determine if it is odd
 n = length(htwt$Weight)
 n

## [1] 20

 odd(n)

## [1] FALSE

 # Since 20 is even we will average the two middle SORTED observations
 n.low = n/2
 n.low

## [1] 10

 n.high = (n/2)+1
 n.high

## [1] 11

 wt.sorted = sort(htwt$Weight)
 wt.sorted

## [1] 82 87 87 101 103 110 112 119 119 122 125 151 155 157 159 190 191 195 199
## [20] 228

 wt.sorted[n.low]

## [1] 122

 wt.sorted[n.high]

## [1] 125

 wt.median = (wt.sorted[n.low]+wt.sorted[n.high])/2
 wt.median

## [1] 123.5

 # Or we can use the internal function median
 median(htwt$Weight)

## [1] 123.5

### The Variance and Standard Deviation

The standard deviation is the almost average distance that the oberved values fall away from the mean.

R has a function ***var*** that returns the variance of the observations. Taking the square root of the variance provides the standard deviation.

 # The mean and number of observations were computed above
 n

## [1] 20

 mean(htwt$Weight)

## [1] 139.6

 # We subtract the mean from all of the observations to get the differences, d
 wt.d = htwt$Weight - mean(htwt$Weight)
 wt.d

## [1] 19.4 15.4 17.4 -14.6 -36.6 -17.6 -38.6 -57.6 88.4 59.4 55.4 -29.6
## [13] 51.4 11.4 -20.6 -20.6 -27.6 -52.6 50.4 -52.6

 # We now square the differences
 wt.d2 = wt.d^2 #x^2 means "x squared"
 wt.d2

## [1] 376.36 237.16 302.76 213.16 1339.56 309.76 1489.96 3317.76 7814.56
## [10] 3528.36 3069.16 876.16 2641.96 129.96 424.36 424.36 761.76 2766.76
## [19] 2540.16 2766.76

 # Now we sum the squared differences
 wt.sumd2 = sum(wt.d2)
 wt.sumd2

## [1] 35330.8

 # The variance is the sum divided by n-1
 wt.var = wt.sumd2 / (n-1)
 wt.var

## [1] 1859.516

 # The internal function var gives the same result
 var(htwt$Weight)

## [1] 1859.516

 # To get the standard deviation we take the square root of the variance
 wt.sd = sqrt(wt.var)
 wt.sd

## [1] 43.1221

 # Using internal functions
 sqrt(var(htwt$Weight))

## [1] 43.1221

A computationally friendlier approach is based upon the sums of the values and their squares.

 # Compute the sum of the observations and the sum of the squared values
 wt.sumx = sum(htwt$Weight)
 wt.sumx

## [1] 2792

 wt.sumx2 = sum(htwt$Weight^2)
 wt.sumx2

## [1] 425094

 # We also use the mean and the number of observations
 n = length(htwt$Weight)
 n

## [1] 20

 wt.mean = wt.sumx/n
 wt.mean

## [1] 139.6

 # Use the magic formula to get the variance
 wt.var = (wt.sumx2 - n\*(wt.mean^2))/(n-1)
 wt.var

## [1] 1859.516

 # Using the internal mean function
 (sum(htwt$Weight^2) - n\*(mean(htwt$Weight)^2))/(n-1)

## [1] 1859.516

 # The standared deviation is the square root of the variance
 wt.sd = sqrt(wt.var)
 wt.sd

## [1] 43.1221

 # And the internal functions agree
 var(htwt$Weight)

## [1] 1859.516

 sqrt(var(htwt$Weight))

## [1] 43.1221

### The Interquartile Range (IQR)

We start by recalling that the range is the distance between the largest and smallest values. R’s internal function ***range*** actually provides these values and not the distance between them.

 # To find the largest and smallest values we can cheat and use min and max
 wt.min = min(htwt$Weight)
 wt.min

## [1] 82

 wt.max = max(htwt$Weight)
 wt.max

## [1] 228

 wt.range = wt.max - wt.min
 wt.range

## [1] 146

 # Using R's internal range function is not much easier
 temp = range(htwt$Weight)
 temp

## [1] 82 228

 temp[2]-temp[1]

## [1] 146

This section assumes that you have read the document on the five number summary — the minimum, Q1, median, Q3, and the maximum. The IQR makes use of the difference between quartiles to create a measure of spread based upon medians.

 # We use the definition of median to find the quartiles
 n

## [1] 20

 wt.sorted = sort(htwt$Weight)
 wt.q1 = (wt.sorted[5]+wt.sorted[6])/2
 wt.q3 = (wt.sorted[15]+wt.sorted[16])/2
 c(wt.q1, wt.q3)

## [1] 106.5 174.5

 wt.iqr = wt.q3 - wt.q1
 wt.iqr

## [1] 68

 # Using the internal quantile function gives a slightly different result
 wt.5no = quantile(htwt$Weight)
 wt.5no

## 0% 25% 50% 75% 100%
## 82.00 108.25 123.50 166.75 228.00

 wt.iqr2 = wt.5no[4]-wt.5no[2]
 names(wt.iqr2) = "IQR"
 wt.iqr2

## IQR
## 58.5

A histogram of the observations and the mean and median may help.

 hist(htwt$Weight, xlab="Weight", prob=TRUE, col="gray", main="")
 lines(density(htwt$Weight, adjust=2), lty="dotted")
 abline(v=mean(htwt$Weight), col="red")
 abline(v=median(htwt$Weight), col="green")



Repeating this for different types of data shows the effects of skew.

 # A symmetric disribution
 x = rnorm(1000, 12, 3)
 c(mean(x), median(x))

## [1] 11.85041 11.89482

 hist(x, prob=TRUE, col="gray", main="")
 lines(density(x, adjust=2), lty="dotted")
 abline(v=mean(x), col="red")
 abline(v=median(x), col="green")



 # A right skewed disribution
 x = rbeta(1000, 2, 6)
 c(mean(x), median(x))

## [1] 0.2406470 0.2219513

 hist(x, prob=TRUE, col="gray", main="")
 lines(density(x, adjust=2), lty="dotted")
 abline(v=mean(x), col="red")
 abline(v=median(x), col="green")



 # A left skewed disribution
 x = rbeta(1000, 6, 2)
 c(mean(x), median(x))

## [1] 0.7493679 0.7694405

 hist(x, prob=TRUE, col="gray", main="")
 lines(density(x, adjust=2), lty="dotted")
 abline(v=mean(x), col="red")
 abline(v=median(x), col="green")



## z-Scores

The z-score is a measure of how far individual observations fall away from the mean/middle. It measures this distance in standard deviations. Let z = (x - mu)/sigma Then z is units free. If the distribution of the z’s is bell shaped, z-scores that are less than -2 or greater than 2 are often considered to be extreme.

 wt.z = (htwt$Weight - mean(htwt$Weight))/sqrt(var(htwt$Weight))
 wt.z

## [1] 0.4498853 0.3571254 0.4035054 -0.3385735 -0.8487527 -0.4081434
## [7] -0.8951326 -1.3357419 2.0499928 1.3774838 1.2847240 -0.6864229
## [13] 1.1919641 0.2643656 -0.4777132 -0.4777132 -0.6400430 -1.2197921
## [19] 1.1687742 -1.2197921

 hist(wt.z)
 p\_load(lattice)



 histogram(~wt.z)



 stem(wt.z,2)

##
## The decimal point is at the |
##
## -1 | 322
## -0 | 987655
## -0 | 43
## 0 | 3444
## 0 |
## 1 | 2234
## 1 |
## 2 | 0

 # The z-scores aren't particullarly bell shaped, but we can still look for extremes.
 htwt$Weight[(wt.z<=-2) | (wt.z>=2)]

## [1] 228

The z-scores are not particularly bell shaped. If we pretend that they are, we see that the 2.0499928 z-score that corresponds to a weight of 228 might be suspect. Note that the 1.5\*IQR rule did not think that this one was particularly strange.

### Quantile plots

A better way to check to see if the z-scores have a bell shaped (normal) distribution is by using a quantile plot. When the z-scores come from a normal distribution, the plot will have a number of points plotted along the diagonal line. Typically, the fit in the tails is not very good. The middle of the plot should not show any “strange” trends.

 qqnorm(wt.z)
 qqline(wt.z)



 ht.z = (htwt$Height-mean(htwt$Height))/sqrt(var(htwt$Height))
 qqnorm(ht.z)
 qqline(ht.z)



 qqplot(ht.z,wt.z)



The quantile plot for the ***wt.z*** z-scores shows some unfortunate deviations from the diagonal line in the middle of the plot. These suggest that the z-scores are not normally distributed. On the other hand, the plot for the ***ht.z*** z-scores suggests a more normal distribution. If we pretend that they are, we see that the 2.0021022 z-score that corresponds to a height of 79 might be suspect.