# Descriptive Statistics

*Irfan Kanat*

*July 3, 2017*

First order of business in any analytics task is to understand what we have in our hands. We are familiar with some of these methods and others we will learn as we go.

## Visual Inspection

Let us think of an example. Let's say you have 30 bags full of oranges, while there is some difference in weights they are more or less the same.

Don't worry about the code below, I will create a series of weights for our hypothetical bags. Right now the things I do may be hard to follow, but they are not essential anyway.

```
# Setting the seed for random number generator so that examples stay consistent.
set.seed(2017)
# Create a data frame
bagsOfOranges <- data.frame(bagNo = 1:30,
                            weight = rnorm(30, mean = 1.9, sd = .1))
```

Let us say the weights of each bag is stored in bagsOfOranges data.frame. Let us take a look at the data. View command presents a spread sheet like view of the data.

```
# Or you can double click on the name of the data.frame in environment pane in Panel C.
View(bagsOfOranges)
```

```
## Warning in View(bagsOfOranges): X cannot set locale modifiers
```

You can also achieve the same in the commandline by simply typing the name of the dataset.

```
bagsOfOranges
```

```
##    bagNo   weight
## 1      1 2.043420
## 2      2 1.892271
## 3      3 1.973914
## 4      4 1.724140
## 5      5 1.893017
## 6      6 1.945191
## 7      7 1.704163
## 8      8 1.899848
## 9      9 1.873466
## 10    10 2.056322
## 11    11 1.934277
## 12    12 2.057243
## 13    13 1.825327
## 14    14 1.930665
## 15    15 1.756951
## 16    16 2.019443
## 17    17 1.851793
## 18    18 2.031786
## 19    19 1.787017
## 20    20 1.807365
## 21    21 1.914071
## 22    22 1.814777
## 23    23 2.090340
## 24    24 1.735181
## 25    25 1.975962
## 26    26 1.906229
## 27    27 1.861097
## 28    28 1.963667
## 29    29 1.813221
## 30    30 1.909873
```

If you want to simply see top observations you can use head() function.

```
# n parameter specifies how many lines to display, it is optional meaning you can leave it out.
head(bagsOfOranges, n = 4)
```

```
##   bagNo   weight
## 1     1 2.043420
## 2     2 1.892271
## 3     3 1.973914
## 4     4 1.724140
```

Similarly with the bottom observations

```
tail(bagsOfOranges) # here I leave n parameter out, the default is 6 rows.
```

```
##    bagNo   weight
## 25    25 1.975962
## 26    26 1.906229
## 27    27 1.861097
## 28    28 1.963667
## 29    29 1.813221
## 30    30 1.909873
```

If you recall Introduction to Statistical Computing Environments module's, Introduction learning activity, we discussed indexes as a way to access data in a data.frame.

Let us say we want to see only the weight column, there are a few ways to do that.

```
# Method 1
bagsOfOranges[, 2] # second column, notice the position of comma.
```

```
##  [1] 2.043420 1.892271 1.973914 1.724140 1.893017 1.945191 1.704163
##  [8] 1.899848 1.873466 2.056322 1.934277 2.057243 1.825327 1.930665
## [15] 1.756951 2.019443 1.851793 2.031786 1.787017 1.807365 1.914071
## [22] 1.814777 2.090340 1.735181 1.975962 1.906229 1.861097 1.963667
## [29] 1.813221 1.909873
```

```
# Method 2
bagsOfOranges[, "weight"] # with column name
```

```
##  [1] 2.043420 1.892271 1.973914 1.724140 1.893017 1.945191 1.704163
##  [8] 1.899848 1.873466 2.056322 1.934277 2.057243 1.825327 1.930665
## [15] 1.756951 2.019443 1.851793 2.031786 1.787017 1.807365 1.914071
## [22] 1.814777 2.090340 1.735181 1.975962 1.906229 1.861097 1.963667
## [29] 1.813221 1.909873
```

```
# Method 3
bagsOfOranges$weight # with $ operator and column name
```

```
##  [1] 2.043420 1.892271 1.973914 1.724140 1.893017 1.945191 1.704163
##  [8] 1.899848 1.873466 2.056322 1.934277 2.057243 1.825327 1.930665
## [15] 1.756951 2.019443 1.851793 2.031786 1.787017 1.807365 1.914071
## [22] 1.814777 2.090340 1.735181 1.975962 1.906229 1.861097 1.963667
## [29] 1.813221 1.909873
```

Remember we can also index rows. Below are a few ways of doing the same

```
bagsOfOranges[2, ] # Second observation, notice the position of comma
```

```
##   bagNo   weight
## 2     2 1.892271
```

You can request a range of observations. Let us say you are interested in observations 4 through 8.

```
# Method 1
# Observe what : operator creates
4:8
```

```
## [1] 4 5 6 7 8
```

```
# If I use the : operator to index a data.frame, we get what we want.
bagsOfOranges[4:8, ] # Observations 4 through 8
```

```
##   bagNo   weight
## 4     4 1.724140
## 5     5 1.893017
## 6     6 1.945191
## 7     7 1.704163
## 8     8 1.899848
```

```
# Method 2
# Another way it to specify columns explicitly. We can use c() function to combine values
# Observe what c function creates
c(4, 5, 6, 7, 8)
```

```
## [1] 4 5 6 7 8
```

```
# If I use the c operator, I can request specific rows
bagsOfOranges[c(4, 5, 6, 7, 8), ] # Observations 4 through 8
```

```
##   bagNo   weight
## 4     4 1.724140
## 5     5 1.893017
## 6     6 1.945191
## 7     7 1.704163
## 8     8 1.899848
```

```
# Method 3
# Of course you can combine the two approaches
bagsOfOranges[c(4:6, 7, 8), ]
```

```
##   bagNo   weight
## 4     4 1.724140
## 5     5 1.893017
## 6     6 1.945191
## 7     7 1.704163
## 8     8 1.899848
```

Combine function is more useful if you want to get specific observations only. Let us say 3$^{\text{rd}}$ and 7$^{\text{th}}$.

```
bagsOfOranges[c(3, 7), ]
```

```
##   bagNo   weight
## 3     3 1.973914
## 7     7 1.704163
```

Think about what we just learned. This way you can see how your data looks and see specific parts of it.

Let me tell you about a few useful functions, that are helpful in knowing more about your data.

Number of observations, use nrow() function for data.frame and length() function for vectors (each row is a vector).

```
nrow(bagsOfOranges) # bagsOfOranges is a data.frame
```

```
## [1] 30
```

```
length(bagsOfOranges$bagNo) # bagNo is a vector
```

```
## [1] 30
```

Let's revisit filtering data frame with a logic operator (learning activity 3 module 1).

Remember we can use logic operators to evaluate a statement. Let us see which bags have more than 2 pounds of oranges.

```
# Observe the output of logic operator, we are trying to see which bags have more than 2 pounds
bagsOfOranges$weight > 2 # Is weight greater than 2?
```

```
##  [1]  TRUE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE  TRUE FALSE
## [12]  TRUE FALSE FALSE FALSE  TRUE FALSE  TRUE FALSE FALSE FALSE FALSE
## [23]  TRUE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
```

```
# The output is a series of True False
# If we feed this output into out dataframe index, it will only spit out the rows where evaluation is T
bagsOfOranges[bagsOfOranges$weight > 2, ]
```

```
##    bagNo   weight
## 1      1 2.043420
## 10    10 2.056322
## 12    12 2.057243
## 16    16 2.019443
## 18    18 2.031786
## 23    23 2.090340
```

Now a simple exercise for you. Can you make R spit out how many bags have more than 2 pounds of oranges? What is the percentage of overweight bags? (See the end of document for solution)

## Average - Mean

If anyone asks you how much a bag of oranges weights, what would you say?

You will give a figure based on your experience with the bags. This figure will most probably be the mean, or the average weight of bags. You can think the average as the simplest model one can fit the data. If you know next to nothing about other variables, the best you can do in a pinch is to report mean.

In R you can get the average with the mean() function.

```r
mean(bagsOfOranges$weight)
```

```
## [1] 1.899735
```

## Standard Deviation

I will now create a second batch of bags full of apples. Let us say the apples are not as uniform as oranges and they differ greatly in terms of weights. Don't worry much about this piece of code.

```r
# Create a data frame
bagsOfApples <- data.frame(bagNo = 1:30,
                           weight = rnorm(30, mean = 2.1, sd = .5))
```

Now let us compare apples to oranges in terms of average weight.

```r
mean(bagsOfApples$weight)
```

```
## [1] 2.167115
```

```r
mean(bagsOfOranges$weight)
```

```
## [1] 1.899735
```

The two figures are pretty close to each other. Can we say the a bag of apple will weigh more or less the same as a bag of oranges?

Not really. Mean is a simple model and is useful if the difference (variance) in bags is also similar between the groups. If there is great difference between weight of bags between two groups, mean is not very useful (as one bag can be .7 pounds and the next may be 3.4 pounds).

Let us learn how to figure out the difference. A common measure of difference is standard deviation (sd).

Let us compare apples to oranges in terms of standard deviation with sd() function.

```r
sd(bagsOfApples$weight)
```

```
## [1] 0.5588494
```

```r
sd(bagsOfOranges$weight)
```

```
## [1] 0.1051508
```

You can see the difference between weights of Orange bags is much smaller than the difference between the weights of Apples.

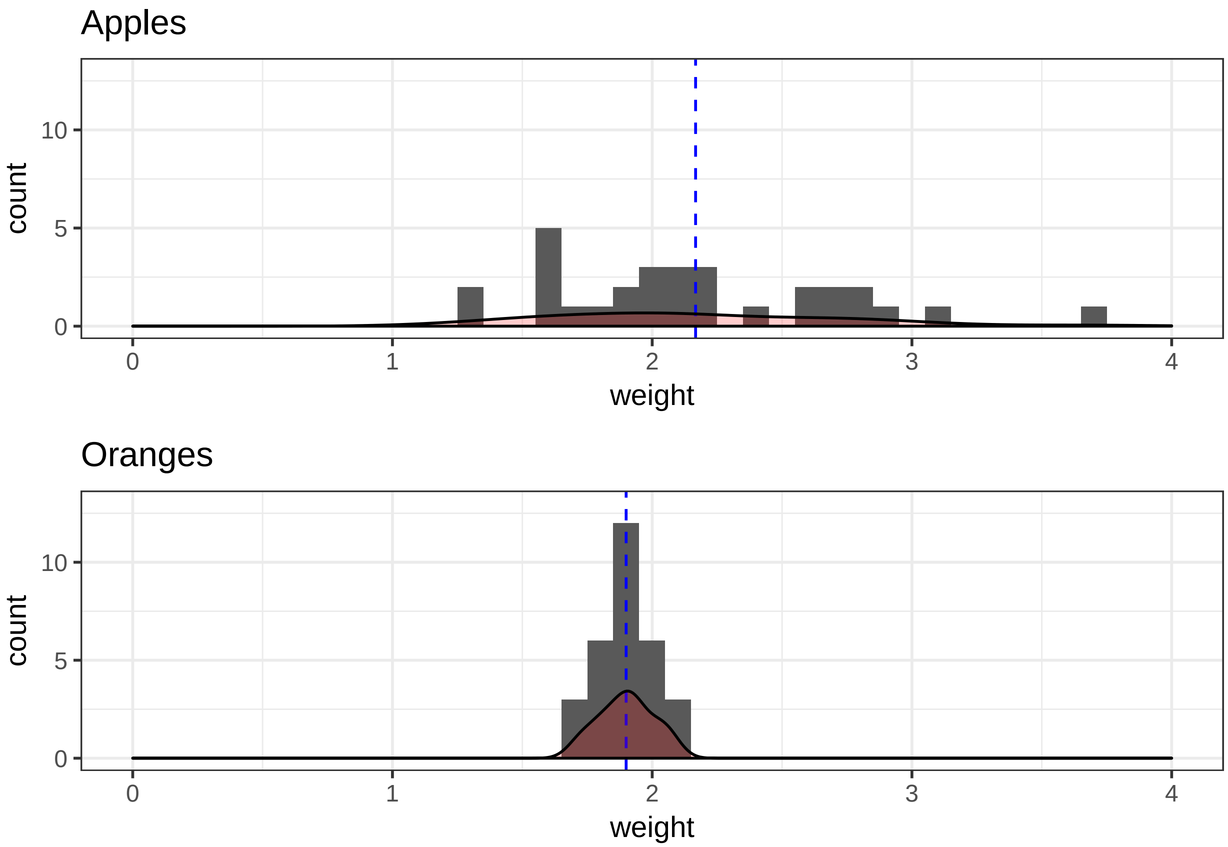
Let me take a segway and show you what this difference means visually. For now do not worry too much about the code, you will learn how to visualize data in the following learning activity.

```r
library(ggplot2) # Load the necessary library for fancy plots
library(gridExtra) # The library to put two panels with plots together

# Create the plot for Apples
p1 <- ggplot(data = bagsOfApples, aes(weight)) + # initialize a plot for data
  xlim(0, 4) + # Set the limits of x axis to keep plots comparable
  ylim(0, 13) + # Set the limits of y axis to keep plots comparable
  ggtitle("Apples") + # Set the title to be able to identify plots
  geom_histogram(binwidth = .1) + # Set the type of plot as a histogram
  # Add the mean into the plot as a blue dashed line
  geom_vline(data = bagsOfApples, aes(xintercept = mean(weight)),
             linetype = "dashed", colour = "blue") +
  # Overlay density plot to show distribution
  geom_density(alpha = .2, fill = "red") +
  theme_bw() # Make it look good

# Create the plot for Oranges
p2 <- ggplot(data = bagsOfOranges, aes(weight)) + # initialize a plot for data
  xlim(0, 4) + # Set the limits of x axis to keep plots comparable
  ylim(0, 13) + # Set the limits of y axis to keep plots comparable
  ggtitle("Oranges") + # Set the title to be able to identify plots
  geom_histogram(binwidth = .1) +  # Set the type as histogram
  # Add the mean into plot as a blue dashed line
  geom_vline(data = bagsOfOranges, aes(xintercept = mean(weight)),
           linetype = "dashed", colour = "blue") +
  # Overlay density plot to show distribution
  geom_density(alpha = .2, fill = "red") +
  theme_bw() # Make it look good

grid.arrange(p1, p2)
```



Look at the above plots. These are called histograms, the histogram shows us how many bags were observed in a certain weight class.

Oranges is pretty normally distributed with a small standard deviation, whereas apples have a large standard deviation. Hence the mean value does not quite work to compare apples and oranges as they are quite differently distributed.

### T Test

I want to take a tiny segway here. If you want to compare two groups and to see if they are the same or not (Do men and women weight the same? Are business students and engineering students have similar SAT scores?) you can use a t test. In a normal distribution roughly 99.7% of observations will be 2 standard deviations away from the mean. Basically a T test compares the mean values of each group and using standard deviations, it determines if they are indeed the same.

```
t.test(bagsOfApples$weight, bagsOfOranges$weight)
```

```
##
##  Welch Two Sample t-test
##
## data:  bagsOfApples$weight and bagsOfOranges$weight
## t = 2.5754, df = 31.051, p-value = 0.015
## alternative hypothesis: true difference in means is not equal to 0
## 95 percent confidence interval:
##  0.05564843 0.47911249
## sample estimates:
## mean of x mean of y
##  2.167115  1.899735
```

The p value below .05 indicates the t test shows statistically significant differences between bags of apples and bags of oranges.

## Median

The apples and oranges were normally distributed (a bell curve with more or less equal number of observations on either side of mean). When distribution is not normal (Consider how wages are distributes with lots of observations on the left with low wages and few very high observations to the right) it is best to report the median instead of the mean. Median is the middle observation, with exactly the same number of observations above and below median.

Since apples were normally distributed median will be similar to mean.

```
median(bagsOfApples$weight)
```

```
## [1] 2.100625
```

## Summary Statistics

It is all good and fine to evaluate variables one by one, but realistically you won't have the time to go through a data set, evaluating variables one by one. There may be hundreds of variables. Let us talk about how to get summary statistics in R easily.

Let us load a dataset to further explore. R and R packages often come with useful data. Let us use a prepackaged dataset here.

In this document we will analyze the Motor Trends data. The dataset was compiled from 1974 issues of Motor Trends magazine and is included with R Base package.

Let us start with loading the dataset.

```
data(mtcars)
```

As we learned in the section on packages, you can querry the documentation for almost anything. Including the datasets included in packages. The document includes descriptions of the variables.

```
?mtcars
```

### Summary Function

Easiest way is to use summary() function in R. It comes bundled with R Base and is loaded in memory by default, so you don't have to load any libraries.

```
# A summary of variables
summary(mtcars)
```

```
##       mpg             cyl             disp             hp       
##  Min.   :10.40   Min.   :4.000   Min.   : 71.1   Min.   : 52.0  
##  1st Qu.:15.43   1st Qu.:4.000   1st Qu.:120.8   1st Qu.: 96.5  
##  Median :19.20   Median :6.000   Median :196.3   Median :123.0  
##  Mean   :20.09   Mean   :6.188   Mean   :230.7   Mean   :146.7  
##  3rd Qu.:22.80   3rd Qu.:8.000   3rd Qu.:326.0   3rd Qu.:180.0  
##  Max.   :33.90   Max.   :8.000   Max.   :472.0   Max.   :335.0  
##       drat             wt             qsec             vs        
##  Min.   :2.760   Min.   :1.513   Min.   :14.50   Min.   :0.0000  
##  1st Qu.:3.080   1st Qu.:2.581   1st Qu.:16.89   1st Qu.:0.0000  
##  Median :3.695   Median :3.325   Median :17.71   Median :0.0000  
##  Mean   :3.597   Mean   :3.217   Mean   :17.85   Mean   :0.4375  
##  3rd Qu.:3.920   3rd Qu.:3.610   3rd Qu.:18.90   3rd Qu.:1.0000  
##  Max.   :4.930   Max.   :5.424   Max.   :22.90   Max.   :1.0000  
##        am              gear            carb      
##  Min.   :0.0000   Min.   :3.000   Min.   :1.000  
##  1st Qu.:0.0000   1st Qu.:3.000   1st Qu.:2.000  
##  Median :0.0000   Median :4.000   Median :2.000  
##  Mean   :0.4062   Mean   :3.688   Mean   :2.812  
##  3rd Qu.:1.0000   3rd Qu.:4.000   3rd Qu.:4.000  
##  Max.   :1.0000   Max.   :5.000   Max.   :8.000  
```

The output from summary function displays Mean and Median as well as quartiles, minimum and maximum. This output is very useful in trying to guess the underlying distribution of the dataset.

## Describe Function

A more detailed output can be obtained by the describe function in psych package (you know how to install and activate packages from previous module).

```
library(psych)
```

```
## 
## Attaching package: 'psych'
```

```
## The following objects are masked from 'package:ggplot2':
## 
##     %+%, alpha
```

```
describe(mtcars)
```

```
##      vars  n   mean     sd median trimmed    mad   min    max  range  skew
## mpg     1 32  20.09   6.03  19.20   19.70   5.41 10.40  33.90  23.50  0.61
## cyl     2 32   6.19   1.79   6.00    6.23   2.97  4.00   8.00   4.00 -0.17
## disp    3 32 230.72 123.94 196.30  222.52 140.48 71.10 472.00 400.90  0.38
## hp      4 32 146.69  68.56 123.00  141.19  77.10 52.00 335.00 283.00  0.73
## drat    5 32   3.60   0.53   3.70    3.58   0.70  2.76   4.93   2.17  0.27
## wt      6 32   3.22   0.98   3.33    3.15   0.77  1.51   5.42   3.91  0.42
## qsec    7 32  17.85   1.79  17.71   17.83   1.42 14.50  22.90   8.40  0.37
## vs      8 32   0.44   0.50   0.00    0.42   0.00  0.00   1.00   1.00  0.24
## am      9 32   0.41   0.50   0.00    0.38   0.00  0.00   1.00   1.00  0.36
## gear   10 32   3.69   0.74   4.00    3.62   1.48  3.00   5.00   2.00  0.53
## carb   11 32   2.81   1.62   2.00    2.65   1.48  1.00   8.00   7.00  1.05
##      kurtosis    se
## mpg     -0.37  1.07
## cyl     -1.76  0.32
## disp    -1.21 21.91
## hp      -0.14 12.12
## drat    -0.71  0.09
## wt      -0.02  0.17
## qsec     0.34  0.32
## vs      -2.00  0.09
## am      -1.92  0.09
## gear    -1.07  0.13
## carb     1.26  0.29
```

As you can see, beyond what is reported in summary, describe function also reports standard deviation and standard error.

You may notice skewness and kurtosis of the data. These are useful in estimating how far the distribution is from normal (bell curve). Skewness shows how far from symetrical the distribution. Skewness for a normal distribution is zero, symetrical data should have values close to zero. Kurtosis is the amount of data below the tails. The kurtosis for normal distribution is 0. The further from the normal, the more different the values will be.

At first the skewness and kurtosis may seem alarming, I just want you to realize there is no normal distribution in real life. All data differs from normal to some degree. You can test for normality using various tests like Kolmogorov-Smirnov tests. If your goal is prediction, deviations from normalcy is not a really big deal. The real problems start when you try using the model for explanation.

## Correlations

Correlation is a good measure of how two variables are related. It basically shows you what happens to one variable when you change another. This is a bivariate comparison, meaning it will ignore the effect of all other variables besides the two.

To obtain a correlation table in R you simply use cor() function.

```
# Correlation table for first 4 variables (due to space concerns)
cor(mtcars[, 1:4])
```

```
##             mpg        cyl       disp         hp
## mpg   1.0000000 -0.8521620 -0.8475514 -0.7761684
## cyl  -0.8521620  1.0000000  0.9020329  0.8324475
## disp -0.8475514  0.9020329  1.0000000  0.7909486
## hp   -0.7761684  0.8324475  0.7909486  1.0000000
```

A variable will always be perfectly correlated with itself (perfect correlation = 1), if there is no correlation the value will be 0. If the correlations are negative, this means variables are inversely related.

You can see that amount of horse power in a car is inversely related to miles per galon (-0.776), but the number of cylinders is positively related to horsepower.

## Tabulating the Data

If you want to see how various groups (groups often mean categorical variables) compare to each other, you can always tabulate the data with table command.

At its most basic you can get a count of each category for a single categorical variable. Let us tabulate the number of gears.

```
table(mtcars$gear)
```

```
##
##  3  4  5
## 15 12  5
```

There are 15 cars with 3 gears and 5 cars with 5 gears.

Let us see a tabulation of Transmission type (Automatic = 0 and Manual = 1) by Number of Cylinders.

```
# bivariate comparisons of categorical variables
table(mtcars[, c("am", "cyl")])
```

```
##    cyl
## am   4  6  8
##   0  3  4 12
##   1  8  3  2
```

Based on the data most cars with 8 cylinders come with automatic transmissions. For 4 cylinder cars, most of them are manual.

## Solutions to Exercises

1 - Can you make R spit out how many bags have more than 2 pounds of oranges? What is the percentage of overweight bags?

```
# Use nrow() function to figure out the number
nrow(bagsOfOranges[bagsOfOranges$weight > 2, ])
```

```
## [1] 6
```

```
# We can divide this with the total number of rows to get percentage.
nrow(bagsOfOranges[bagsOfOranges$weight > 2, ]) / nrow(bagsOfOranges)
```

```
## [1] 0.2
```

## Housekeeping

Ignore this part, basically this part is to export data created so it can be used in Rmarkdown documents later.

```
write.csv(bagsOfApples, file = "data/bagsOfApples.csv", row.names = F)
write.csv(bagsOfOranges, file = "data/bagsOfOranges.csv", row.names = F)
```