Descriptive (Incisive) Statistics

Neil J. Hatfield

Last Updated: 2022-08-26

Welcome to the latest installment of using R and RStudio. This guide will be looking at descriptive/incisive statistics. As you look through this guide, attempt to calculate the for yourself using the example code. Do this with both the demo data and a new data set (say from HW #1.1).

Getting Started

To get started we need to do two major tasks: load the necessary packages and load our data.

Load Packages

When you start a new session of R, one of the things that you should do is to load the extra packages that will bring new functions and new capabilities. If you don't load the necessary packages, your code might not work.

For descriptive/incisive statistics, the bulk of the statistics are part of base R. Thus, we don't really need to load any additional packages. However, there are a few packages that provide us with a few more statistics and a few functions that will provide many statistics at once. Additionally, I'll be demonstrating how to make tables of values in R Markdown in this guide.

Thus, we will need to load the following packages: `psych`, `knitr`, and `kableExtra`. Go ahead and load these packages. If you need a refresher on what code does this, check below:

```
packages <- c("tidyverse", "psych", "knitr", "kableExtra")
lapply(
  X = packages,
  FUN = library,
  character.only = TRUE
)
```

Notice that I included `tidyverse` in my package list. I almost always load `tidyverse` as I never know when I might need to manipulate data or make a quick visualization.

Load Data

For this guide I'm going to use two different data sets: the class demo Oreo data set and `palmerpenguins`.

```
oreoData <- read.table(
  file = "https://raw.github.com/neilhatfield/STAT461/master/dataFiles/classDemoOreo.dat",
  header = TRUE,
  sep = ","
)

# If you don't have the palmerpenguins package run the
# following line in your console, removing the # sign
```

```
# install.packages("palmerpenguins")

penguins <- palmerpenguins::penguins
```

We've now loaded the two data sets. I'm going to go head and show the structure of both files, and you'll have to trust that I used the `View` function in my console to look at both data frames.

```
str(oreoData)
```

```
## 'data.frame':    60 obs. of  2 variables:
##  $ Filling.Mass: num  3.27 3.12 3.15 3.27 3.24 ...
##  $ Type        : chr  "Regular" "Regular" "Regular" "Regular" ...
```

```
str(penguins)
```

```
## tibble [344 x 8] (S3: tbl_df/tbl/data.frame)
##  $ species          : Factor w/ 3 levels "Adelie","Chinstrap",..: 1 1 1 1 1 1 1 1 1 1 ...
##  $ island           : Factor w/ 3 levels "Biscoe","Dream",..: 3 3 3 3 3 3 3 3 3 3 ...
##  $ bill_length_mm   : num [1:344] 39.1 39.5 40.3 NA 36.7 39.3 38.9 39.2 34.1 42 ...
##  $ bill_depth_mm    : num [1:344] 18.7 17.4 18 NA 19.3 20.6 17.8 19.6 18.1 20.2 ...
##  $ flipper_length_mm: int [1:344] 181 186 195 NA 193 190 181 195 193 190 ...
##  $ body_mass_g      : int [1:344] 3750 3800 3250 NA 3450 3650 3625 4675 3475 4250 ...
##  $ sex              : Factor w/ 2 levels "female","male": 2 1 1 NA 1 2 1 2 NA NA ...
##  $ year             : int [1:344] 2007 2007 2007 2007 2007 2007 2007 2007 2007 2007 ...
```

Because I'm reloading the Oreo data from the source file, I do need to tell my current session of R that Type should be a factor. In the penguins data, I see that species, island, and sex are already marked as factors. You'll also notice in the preview of values that there are a few `NA`s. `NA` denotes a missing value. We will see what happens in R when you go to use data with missing values.

```
# Quickly recode Type as a factor
oreoData <- oreoData %>%
  dplyr::mutate(
    dplyr::across(where(is_character), as_factor)
  )
```

Descriptive/Incisive Statistics

Now that we have data in, we can use R to calculate the values of various statistics. There are so many statistics out there that if I were to talk about all of them, we would do nothing else for the semester. Instead, I'm going to focus on some of the most useful ones. In this guide I will not get into any statistics whose use is primarily for statistical inference (that will be a different guide).

I will organize the statistics by what attribute of a data collection they measure.

Size

When talking about the size of a data set, we have several meanings:

- How many observations are there?
- How many attributes did we track?
- How many missing/nonmissing values are there?

The first two questions are answered by the `nrow` and `ncol` functions, respectively. `nrow(oreoData)` will reveal that there are 60 observations in the data frame, while `ncol(oreoData)` reveals that there are 2 columns. For the penguin data, what code would you write to verify that there are 344 observations and 8 columns?

Notice that `nrow` did not make any distinctions between any types of cookies or species of penguin. You can restrict the counting provided you know some information. For example, `nrow(penguins[which(penguins$species == "Gentoo"),])` will yield that there are 124 Gentoo penguins. You can get counts by groups a bit more easily using the methods I'll talk about later.

Knowing whether you have missing values, and how many, can be useful. If you have a high proportion of missing values, you might need to rethink your data collection methods and your results might not be good reflections of the broader population.

We can use code such as `nrow(oreoData[is.na(oreoData$Type),])` to get a count of how many observations are missing the Type attribute (0 observations.) Similarly, we can find out that 2 penguins are missing measurements of their flipper length (`nrow(penguins[is.na(penguins$flipper_length_mm),])`).

If you wanted the count of observations that are NOT missing, you can use R's logical negation, `!`. Thus, `nrow(penguins[!is.na(penguins$flipper_length_mm),])` yields how many observations aren't missing (342).

Extrema

If an attribute has an order that indicates that objects/beings have more or less of the attribute, then we can look for the extrema of the collection. This is done through the *sample max[imum]* (`max`) and the *sample min[imum]* (`min`).

Both functions are easy to use `max(oreoData$Filling.Mass)` and `min(oreoData$Filling.Mass)` will yield results of 7.21 and 2.292, respectively. However, we need to have care. Notice what you get if you swap `penguins$flipper_length_mm` for `oreoData$Filling.Mass` in the `max` function: NA. We got NA because there are missing values in the column of flipper length. To correct this, we will add the `na.rm = TRUE` argument to both `max` and `min`. `max(penguins$flipper_length_mm, na.rm = TRUE)` gives 231, and `min(penguins$flipper_length_mm, na.rm = TRUE)` yields 172.

Pretty much every statistic we're going to look at today has the `na.rm` argument. Thus, a good habit to get into is to always include `na.rm = TRUE` to ensure that you get a value.

Middle

The middle of an ordered data collection is measured by the *sample median*, often denoted with $\widetilde{x}$. In R, to get the value of the *sample median* you will use the `median` function. For example, `median(oreoData$Filling.Mass, na.rm = TRUE)` will review the value of the *sample median* of créme filling mass, ignoring type of cookie; 4.569 grams.

Order

The order of a data collection can be measured through a variety of statistics. For example, you can use order statistics to get the rank (smallest to largest) of each data set. To do this you could use the `rank` function on the whole vector (i.e., `rank(oreoData$Filling.Mass)`; you will get a vector of the same length with the ranks for each observation).

More common are **quantiles**. The most common quantiles are the 5 *quartiles* and *percentiles*. For both of these, we will make use the of `quartile` function in R.

Remember that the five quartiles are:

1) The *zeroth quartile*, Q_0, a.k.a, *sample min*
2) The *first quartile*, Q_1
3) The *second quartile*, Q_2, a.k.a., *sample median*, $\widetilde{x}$
4) The *third quartile*, Q_3
5) The *fourth quartile*, Q_4, a.k.a., *sample max*

The five quartiles for the filling mass data are: 2.292, 3.25275, 4.569, 6.41225, 7.21 (`quantile(oreoData$Filling.Mass, na.rm = TRUE)`).

You can adjust what type of quantile you're looking at with the `probs` argument. Suppose I want to look at octiles for the flipper lengths; I would need to use `probs = seq(0, 1, 1/8)` in the `quantile` function to get these nine values. Thus, `quantile(penguins$flipper_length_mm, probs = seq(0, 1, 1/8), na.rm = TRUE)` yields the values 172, 186, 190, 193, 197, 207, 213, 219.375, 231. Notice that I used `1/8` rather than `0.125`; remember that R will do calculations for you.

Group Performance

Group Performance deals with how well a collection did in terms of some goal. Most often, the goals deal with amassing something. There are two statistics which measure this that are worth mentioning: *GP+* and *GP**. The key between them deals with whether your data values may be meaningfully combined with addition or if multiplication is the way to go.

If your values are additive in nature, then you can use the `sum` function in R; `sum(penguins$body_mass_g, na.rm = TRUE)` yields the total amassed mass of the penguins (1437000 grams).

If your values are multiplicative in nature, then you should use the `prod` function in R. This will multiply all of the values together. The data in my two example data frame are all additive in nature (most things are). However, here are some examples where there is a multiplicative structure to the data:

- Rates
- Percentages
- pH measurements
- Anything on a logarithmic scale

Adjusted Group Performance

The group performance attribute of a data collection is only useful if 1) you have no plans to compare the collection to any other collection and you don't want to connect to a bigger population or 2) your only comparisons are with groups the exact same size as each other and you don't want to connect to a bigger population. If you want to make connections to the population or make comparisons between potentially different sizes of collections, then you need to look at the Adjusted Group Performance attribute. The adjustment accounts for the size of your collection. The statistics which measure this attribute are collectively known as *sample means*. This family includes all of the following:

- *Sample Arithmetic Mean* (*SAM*)
- *Sample Geometric Mean* (*SGM*)
- *Sample Harmonic Mean* (*SHM*)
- *Sample Contra-Harmonic Mean*
- *Sample 1st Contra-Geometric Mean*
- *Sample 2nd Contra-Geometric Mean*
- *Nicomachus' Mean*
- *Pappus's Mean*
- *Heron's Mean*
- *Archimedean Mean*
- *Euclidean Mean*
- *Logarithmic Mean*
- *Huntington Mean*
- *Gauss' Arithmetic-Geometric Mean*
- And more

Thus, you need to have care with which statistic you're using. Just saying "the value of the *sample mean* is..." is **too vague**; which mean do you mean? I will not go into depth for all of the means, but I will briefly discuss the three most commonly used means.

Sample Arithmetic Mean

The *Sample Arithmetic Mean* or *SAM* is the most commonly used *sample mean*. This statistic has a critical assumption that most people fail to realize: the input data set must consist of values which can be meaningfully combined through addition. If this assumption is not satisfied, then the resulting value of the *SAM* is junk at best, misleading at worst.

In R, you'll use the `mean` function to get values of the *SAM*. (You may add on the argument `trim` to get a *trimmed [arithmetic] mean*.) If we wanted to know the value of the *SAM* for the flipper lengths, we would use `mean(penguins$flipper_length_mm, na.rm = TRUE)` and find a value of 200.9152047 mm/penguin. Notice that the output is not rounded. This is a good example that you'll want to use the `round` function (see the Getting Started guide) to make sure numbers look nice. There is little reason to ever present more than four (4) decimals.

Sample Geometric Mean

The *Sample Geometric Mean* or *SGM* is useful when data meet the assumption that the values may be meaningfully combined through multiplication. Examples here include percentages, growth factors, many rates, and anything best described by a logarithmic scale.

To have R calculate values of the *SGM* you will need to load th `psych` package and then use the `geometric.mean` function. For this example, let's say that $\mathcal{A}$ consists of a set of annual interest rates for a non-fixed rate student loan; $\mathcal{A} = \{6.8, 7.2, 6.8, 5.2, 5.4.7.8\}$. These values may not be meaningfully combined through addition, but may with multiplication. The value of the *SGM* for this collection is 6.4635881, obtained with the code `psych::geometric.mean(c(6.8, 7.2, 6.8, 5.2, 5.4, 7.8), na.rm = TRUE)`. Notice that you can directly enter values as I did with the `c(6.8, 7.2, 6.8, 5.2, 5.4, 7.8)`; the `c` function is for combine/concatenate and will turn all of the inputs into a vector.

An alternate method is to use the `DescTools` package's `Gmean` function.

Sample Harmonic Mean

The *Sample Harmonic Mean* or *SHM* is a useful statistic when you are dealing with rates and ratios, harmonics, and optics. The underlying assumption here is that there is an additive structure for the *multiplicative inverses* (i.e., $\frac{1}{x_i}$). Notice that adds on the additional assumption that no data values are zero; $x_i \neq 0$.

You will also need to use the `psych` package to calculate the *SHM* by calling `harmonic.mean`. To ensure that your data free of zeros, you can use the argument `zero = FALSE` which will convert any zeros in `NA`. The `NA`'s will then be ignored with our standard `na.rm = TRUE`.

Just as with the *SGM*, you can also use the `DescTools` package's `Hmean` function.

When Data Are Rates

Rates are special types of numbers. Knowing a bit about how the data were collected and the rates constructed will help you decide which of the above *sample means* is most appropriate. To help you get a sense of what to look for, I'll use the example of speed—the rate (of change) of distance traveled with respect to the time spent traveling, which I'll represent with X_i.

- If everyone traveled for the same amount of time BUT could go any distance in that time, then use the *SAM*.
  - $x_i = \frac{\text{any distance}}{\text{same amount of time}}$ use *SAM*
- If everyone traveled the same amount of distance BUT could do so in any amount of time, then use the *SHM*.
  - $x_i = \frac{\text{same distance}}{\text{any amount of time}}$ use *SHM*
- If everyone could travel for however much time and for whatever distance they wanted, then use the *SGM*.
  - $x_i = \frac{\text{any distance}}{\text{any amount of time}}$ use *SGM*

Variation as Spread

There are two principle statistics for measuring Variation as Spread (as opposed to Variation as Deviation or Variation as Consistency): the *sample range* and the *interquartile range* or *IQR*.

While there is a function in R called `range`, this function is not a statistic, and instead returns the values of the *sample min* and *sample max* as a vector. However, you can quickly calculate the the value of the *sample range* yourself be remembering that this statistic just measures the distance between the two extrema. Thus, since R will perform calculations for you, you just need to use `max(oreoData$Filling.Mass, na.rm = TRUE) - min(oreoData$Filling.Mass, na.rm = TRUE)` to get the value; 4.918.

For the *interquartile range* (*IQR*), you can use the `IQR` function. Thus, `IQR(oreoData$Filling.Mass, na.rm = TRUE)` will tell us that the interval going from Q_1 to Q_3 is 3.1595 grams wide.

Variation as Deviation

Measuring Variation as Deviation has resulted in just as many, if not more, statistics as measuring Adjusted Group Performance. Just as there are the *Sample Arithmetic Mean*, *Sample Geometric Mean*, and *Sample Harmonic Mean*, there are matching *sample variance* statistics: *Sample Arithmetic Variance* (SAV), *Sample Geometric Variance* (SGV), and *Sample Harmonic Variance* (SHV). In addition to the *sample variances*, there are also *sample standard deviations*.

The `DescTools` package does have the `Gsd` function for calculating the value of the *Sample Geometric Standard Deviation*.

Sample Variances and *Sample Standard Deviations*

To get the value of the the *Sample Arithmetic Variance* (SAV), you will use a call such as `var(oreoData$Filling.Mass, na.rm = TRUE)`. You would find the result to be 2.7515856 sq. grams. Remember, many squared versions of units of measure are just weird to think about. Thus, the *Sample Arithmetic Standard Deviations* ($SASD$) will give more familiar units. You could get these values through an application of the `sqrt` function OR you could just use the `sd` function: `sd(oreoData$Filling.Mass, na.rm = TRUE)` yields 1.6587904 grams. Again, notice that using the `round` function would be wise here.

Median Absolute Deviation (MAD)

If you need a more robust and resistant statistic to measure Variation as Deviation, you might want to consider the *Median Absolute Deviation* or *MAD*. The *MAD* has the nice benefit of not inflating the measure as much as the SAV does.

To use this statistic, you simply use `mad(oreoData$Filling.Mass, na.rm = TRUE)` to get the value; in our case 2.1683025 grams.

Skewness and Kurtosis

Keeping in mind that the *sample means* and *sample variances* are often the top estimators for the first moment and second central moment, respectively. However, examining estimates of the third and fourth central (and standardized) moments are useful for making determinations about whether data follow a Gaussian ("normal") distribution. The *Sample Skewness* and *Sample Excess Kurtosis* statistics provide these estimates, respectively.

You will need to use the `psych` package for both of these statistics. To get the value of *Sample Skewness* of a data set, you will need to do the following `psych::skew(oreoData$Filling.Mass, na.rm = TRUE)` (value is 0.0284702). For *Sample Excess Kurtosis* you use this code (**look at the spelling**): `psych::kurtosi(oreoData$Filling.Mass, na.rm = TRUE)` (yields a value of -1.9319789).

Note: the statistic *Sample Excess Kurtosis* does not provide a direct estimate of the fourth central-standardized moment; but rather the difference from the Gaussian distribution's Kurtosis.

Measures of Association

If you want a measure of the association, there are also many different statistics you may use. In R, the `cor` function will produce the values of Pearson's correlation, Spearman's correlation, or Kendall's τ (tau) for your selected data. To choose which you'll use the `method` argument along with one of `"pearson"`, `"spearman"`, or `"kendall"`.

Instead of `na.rm = TRUE`, you will need to use either of the following: `use = "complete.obs"` or `use = "pairwise.complete.obs"`. The former case is when you are only putting in two data vectors; the later for when you are putting into two data frames ***with only numeric data columns*** and want the correlations between all combinations.

For example, suppose I wanted to get the value of Pearson's Correlation for penguin bill length and body mass. To do so, I would need to code `cor(penguins$bill_length_mm, penguins$body_mass_g, method = "pearson", use = "complete.obs")`; this yields a value of 0.5951098.

Most Frequent Value

You'll notice that I haven't mentioned anything about the the sample mode (i.e., the most frequently occurring value). After all, this is the third thing that students often learn about ("mean, median, mode"). The reason is quite simple: the mode is

NOT a statistic. Recall that statistic is a function and functions may only return 1 output for each unique input. Consider the data set $\mathcal{A} = \{1, 2, 3, 3, 3, 2, 2, 4\}$. Both 2 and 3 occur the same number of times, and more than any other value. Thus, both are the "mode" of the sets. However, this means that there are TWO outputs, which violate the definition of a function. (And no, you can't count "(2, 3)" as a single entity; non-multivariate statistics are functions that map to the Real number line not to a two dimensional space.) Base R does not provide any functions to calculate the sample mode; the `mode` function that exists deals with how R stores an object.

Getting Multiple Values at Once

Rarely should you ever rely on just one statistic to describe your data. A much better practice is to use a variety of statistics so that you can build a deeper and richer understanding of the data collection. In a command line driven package like R, this means you have to write a lot of code. Unless, that is, someone has already done the work for you.

If you just want Tukey's Five Number Summary (the five *quartiles*) and the value of the *SAM*, then you can use the `summary` function.

```
summary(oreoData$Filling.Mass, na.rm = TRUE)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   2.292   3.253   4.569   4.830   6.412   7.210
```

The `psych` package has a function called `describe` that goes beyond what `summary` can do. This function takes a data vector (*safest option) or an entire data frame (* use with *extreme* caution) and will calculate the values of many of the statistics listed above. If you choose to put in an entire data frame, the function will calculate values of these statistics, even for character data. The `describe` function will return a new data frame that contains the requested statistics and their values.

```
summaryOreo <- psych::describe(
  x = oreoData$Filling.Mass,
  na.rm = TRUE,
  skew = TRUE,
  ranges = TRUE,
  quant = c(0.25, 0.75),
  IQR = TRUE
)

row.names(summaryOreo) <- "Filling Mass"

summaryOreo
```

```
##              vars  n mean   sd median trimmed  mad  min  max range skew
## Filling Mass    1 60 4.83 1.66   4.57     4.8 2.17 2.29 7.21  4.92 0.03
##              kurtosis   se  IQR Q0.25 Q0.75
## Filling Mass    -1.93 0.21 3.16  3.25  6.41
```

The `describe` function will yield values of the following statistics when you used the above code: *SAM*, *SASD*, *Sample Median*, a *10% Trimmed SAM* (10% from both ends of ordered data), *MAD*, *Sample Min*, *Sample Max*, *Sample Range*, *Sample Skewness*, *Sample Excess Kurtosis*, *Standard Error* for the *SAM*, *IQR*, *First Quartile*, and *Third Quartile*. Additionally, we have the number of observations. The `vars` column is just the index number and may be safely ignored.

Notice that I assigned the output of `describe` to the object `summaryOreo`. This is so I can manipulate the data frame as well as call values as I need. For example, if I were to use `summaryOreo["Filling Mass", "mean"]` in an inline code chunk, I would be able to display the value of the *SAM*: 4.8304 grams/cookie. A little further on in this guide, I'll show you how to format a table to display your results.

You can customize what statistics get calculated through the various arguments. If you don't want *Sample Skewness* or *Sample Excess Kurtosis*, use `skew = FALSE`. Don't want the value of the *Sample Range*; use `range = FALSE`. Similarly for the *IQR*. If you want different quantiles change the argument to a vector of portions you want. For example `quant = seq(0, 1, 1/8)` will yield all nine octiles. If you don't want any quantiles then use `quant = NULL`

Values of Statistics by Groups

In our course we will almost always want to look at data along any grouping factors. The `psych` package includes an extension to the `describe` that allows you to use a grouping factor: `describeBy`. You'll make use of the same arguments as `describe` plus a few extra:

```
sumGrpOreo <- psych::describeBy(
  x = oreoData$Filling.Mass,
  group = oreoData$Type,
  na.rm = TRUE,
  skew = TRUE,
  ranges = TRUE,
  quant = c(0.25, 0.75),
  IQR = TRUE,
  mat = TRUE,
  digits = 4
)

sumGrpOreo
```

```
##    item      group1 vars  n   mean     sd median trimmed    mad   min   max
## 11    1     Regular    1 30 3.2038 0.1812 3.2515  3.2358 0.0474 2.292 3.323
## 12    2 Double Stuf    1 30 6.4570 0.3022 6.4215  6.4484 0.2587 5.815 7.210
##    range    skew kurtosis     se    IQR  Q0.25  Q0.75
## 11 1.031 -4.2182  18.4250 0.0331 0.0740 3.1992 3.2732
## 12 1.395  0.3234  -0.1999 0.0552 0.3105 6.2865 6.5970
```

The two new arguments are `mat = TRUE` which will allow for a better way to store the results for then displaying them as I will show in the next section. The second is `digits = 4`. This essentially applies the `round` function for you.

Try to come up with the code that will display the penguins' body mass, grouped by species as shown below.

```
##    item    group1 vars   n    mean     sd median trimmed    mad  min  max range
## 11    1    Adelie    1 151 3700.66 458.57   3700 3685.74 444.78 2850 4775  1925
## 12    2 Chinstrap    1  68 3733.09 384.34   3700 3719.64 370.65 2700 4800  2100
## 13    3    Gentoo    1 123 5076.02 504.12   5000 5073.48 555.98 3950 6300  2350
##    skew kurtosis    se   IQR Q0.2 Q0.4 Q0.6 Q0.8
## 11 0.28    -0.63 37.32 650.0 3300 3550 3800 4100
## 12 0.24     0.36 46.61 462.5 3400 3650 3780 4050
## 13 0.07    -0.78 45.45 800.0 4650 4895 5210 5550
```

Displaying Values in Professional Ways

The above displays are useful for when you want to just look at values for yourself. They are not particularly well formatted for sharing with fellow team members, supervisors, or the broader public. For this, you need to present values in a much more professional manner.

Now, you could present the values woven in to the narrative. While this is useful in that doing so will help you to give interpretations of the values, this also results in the inevitable scavenger hunt when you just want to know a particular value. Thus, my recommendation is to prepare a high quality table using the `knitr` and `kableExtra` packages.

Basic Table

The most basic of tables can be generated with the following code:

```
summaryOreo %>%
  knitr::kable()
```

	vars	n	mean	sd	median	trimmed	mad	min	max	range	skew	kurtosis	
Filling Mass	1	60	4.8304	1.65879	4.569	4.800542	2.168302	2.292	7.21	4.918	0.0284702	-1.931979	0.2

While the table does look better than the raw display's above, there is still much we can improve about the display. For instance, we could

- Add a caption/title to the table so that we can reference the table in our narrative,
- Fix the number of digits to reign in extremely long decimals add place commas for large numbers
- Control the alignment of values (left, center, right, etc.)
- Change the names of the columns
- Remove extra columns
- Adjust the styling of the table to improve readbility
- Ensure that the table doesn't run off the page

Improved Tables

We can easily add a caption as well as digits by including two additional arguments: `caption` and `digits`. For `caption` you'll need to include a character string, enclosed in either double or single quotation marks that provides the title/caption for your table. This text should be explain what the table shows in a concise manner. For `digits`, you'll just need to put the number of digits after the decimal place you want. The `big.mark` argument of `format.args = list()` will let you set the character to separate sets of place values (i.e., 123,456 vs. 123456), when needed.

```
# Table 1 Code
summaryOreo %>%
  knitr::kable(
    caption = "Summary Statistics for Oreo Filling Masses",
    digits = 3,
    format.args = list(big.mark = ",")
  ) %>%
  kableExtra::kable_styling(
    latex_options = c("HOLD_position", "scale_down")
  )
```

Table 1: Summary Statistics for Oreo Filling Masses

	vars	n	mean	sd	median	trimmed	mad	min	max	range	skew	kurtosis	se	IQR	Q0.25	Q0.75
Filling Mass	1	60	4.83	1.659	4.569	4.801	2.168	2.292	7.21	4.918	0.028	-1.932	0.214	3.16	3.253	6.412

While Table 1 is an improvement, we can do better. For column alignment and column names, we will need to know how many columns there are AND the order the columns are currently being displayed. We will use the `align` argument which is looking for a series of `'l'` (left), `'c'` (center), and/or `'r'` (right)—one for each column. For the column names, we'll set `col.names` equal to vector that contains the improved names.

```
# Table 2 Code
summaryOreo %>%
  knitr::kable(
    caption = "Summary Statistics for Oreo Filling Masses",
    digits = 3,
    format.args = list(big.mark = ","),
    align = rep('c', ncol(summaryOreo)),
    col.names = c("Vars", "n", "SAM", "SASD", "Median", "Trimmed SAM", "MAD", "Min", "Max",
                  "Range", "Sample Skew", "Sample Ex. Kurtosis", "SE", "IQR", "Q1", "Q3")
  ) %>%
  kableExtra::kable_styling(
    latex_options = c("HOLD_position", "scale_down")
  )
```

Table 2: Summary Statistics for Oreo Filling Masses

	Vars	n	SAM	SASD	Median	Trimmed SAM	MAD	Min	Max	Range	Sample Skew	Sample Ex. Kurtosis	SE	IQR	Q1	Q3
Filling Mass	1	60	4.83	1.659	4.569	4.801	2.168	2.292	7.21	4.918	0.028	-1.932	0.214	3.16	3.253	6.412

I find centered aligned cells (particularly for numbers) to be better as shown in Table 2. Since I know that all of my columns are numeric, I used the `rep` (repeat/replicate) function to create a vector of `'c'`'s that was as long as the number of columns (`ncol(summaryOreo)`). I then reformatted the names of the columns by looking at the display in Table 1.

If you notice, the table is still a bit crowded and we don't necessarily need all of these values. Further, we might like to rearrange them to a more logical order. We can drop and rearrange columns using the `dplyr` package, in particular the `select` function.

```
# Table 3 Code
summaryOreo %>%
  dplyr::select(
    n, min, Q0.25, median, Q0.75, max, mad, mean, sd, skew, kurtosis
  ) %>%
  knitr::kable(
    caption = "Summary Statistics for Oreo Filling Masses",
    digits = 3,
    format.args = list(big.mark = ","),
    align = rep('c', 11),
    col.names = c("n", "Min", "Q1", "Median", "Q3", "Max", "MAD", "SAM", "SASD", "Sample Skew",
                  "Sample Ex. Kurtosis")
  ) %>%
  kableExtra::kable_styling(
    latex_options = c("HOLD_position", "scale_down")
  )
```

Table 3: Summary Statistics for Oreo Filling Masses

	n	Min	Q1	Median	Q3	Max	MAD	SAM	SASD	Sample Skew	Sample Ex. Kurtosis
Filling Mass	60	2.292	3.253	4.569	6.412	7.21	2.168	4.83	1.659	0.028	-1.932

Table 3 is nearly there. If you look through the preceding tables, you'll notice that they are all wider than the text of the page, we should adjust this. Further you'll notice that the tables don't necessarily appear with their code (e.g., Table 3). We can stipulate this control as well.

Professional Looking Tables

What I'm going to present here allows us to create truly professional looking tables in format similar to APA style. To begin with we'll start with the code for Table 3 and then add on some new code.

```
# Table 4 Code
summaryOreo %>%
  dplyr::select(
    n, min, Q0.25, median, Q0.75, max, mad, mean, sd, skew, kurtosis
  ) %>%
  knitr::kable(
    caption = "Summary Statistics for Oreo Filling Masses",
    digits = 3,
    format.args = list(big.mark = ","),
    align = rep('c', 11),
    col.names = c("n", "Min", "Q1", "Median", "Q3", "Max", "MAD", "SAM", "SASD", "Sample Skew",
                  "Sample Ex. Kurtosis"),
    format = "latex",
    booktabs = TRUE
```

```
  )  %>%
  kableExtra::kable_styling(
    font_size = 12,
    latex_options = c("scale_down", "HOLD_position")
  )
```

Table 4: Summary Statistics for Oreo Filling Masses

	n	Min	Q1	Median	Q3	Max	MAD	SAM	SASD	Sample Skew	Sample Ex. Kurtosis
Filling Mass	60	2.292	3.253	4.569	6.412	7.21	2.168	4.83	1.659	0.028	-1.932

Notice that we've added two arguments (`format` and `booktabs`) to the `kable` call. These tell R to be sure to make the table ready for LaTeX production and to get rid of the vertical lines between the columns (`booktabs = TRUE`).

Hopefully you noticed that at the end of each of chunk of code making the tables, I used `kableExtra::kable_styling`. This gives us some additional controls. For example, we can use `font_size` to stipulate what size of font to use in the table. I would not go much smaller than 8 or 9 point font—and only if your table is extremely big. I do not recommend that you use a font size larger than what your narrative is (typically 12pt). The other argument, `latex_options` has a vector of two values at the moment: `"scale_down"` and `"HOLD_position"`. These will make sure that the table fits the width of the page and will appear where you placed the code, respectively. If you have a table with multiple rows, you can include a third value to the vector `"striped"`. This will make a "zebra table" where the rows will alternate between white and grey backgrounds.

Displaying Grouped Summarries and Table Footnotes

The above code works well for relatively simple tables. However, we need to do a bit more work if we want to take the output of `psych::describeBy` and display that as a table. The biggest action is that we need to convert the grouping column (`group1`) into row names. We can use `tibble`'s `column_to_rownames` function. (The `tibble` package is part of `tidyverse`.) Let's see how to make a table similar to Table 4, but using the grouped summaries.

```
# Table 5 Code
sumGrpOreo %>%
  tibble::remove_rownames() %>%
  tibble::column_to_rownames(
    var = "group1"
  ) %>%
  dplyr::select(
    n, min, Q0.25, median, Q0.75, max, mad, mean, sd, skew, kurtosis
  ) %>%
  knitr::kable(
    caption = "Summary Statistics for Oreo Filling Masses",
    digits = 3,
    format.args = list(big.mark = ","),
    align = rep('c', 11),
    col.names = c("n", "Min", "Q1", "Median", "Q3", "Max", "MAD", "SAM", "SASD", "Sample Skew",
                  "Sample Ex. Kurtosis"),
    format = "latex",
    booktabs = TRUE
  )  %>%
  kableExtra::kable_styling(
    font_size = 12,
    latex_options = c("scale_down", "HOLD_position")
  ) %>%
  kableExtra::footnote(
    general = "Here is an example of a footnote.",
    general_title = "Note. ",
    footnote_as_chunk = TRUE
  )
```

Table 5: Summary Statistics for Oreo Filling Masses

	n	Min	Q1	Median	Q3	Max	MAD	SAM	SASD	Sample Skew	Sample Ex. Kurtosis
Regular	30	2.292	3.199	3.252	3.273	3.323	0.047	3.204	0.181	-4.218	18.425
Double Stuf	30	5.815	6.287	6.422	6.597	7.210	0.259	6.457	0.302	0.323	-0.200

Note. Here is an example of a footnote.

The two lines with `tibble` are what converts the `group1` column of `sumGrpOreo` into the row names. The `tibble::remove_rownames()` option will clear out any existing row names; this is a good practice as `column_to_rownames` will **not** replace any existing row names.

The `kableExtra::footnote` call demonstrates how you can add a footnote to your table. In addition to the general footnote (`genral`), you can add numbered notes (`number`), lettered notes (`alphabet`), and/or symbol notes (`symbol`). Check out the help documentation for these.

Summarize with `dplyr`

In addition to using `psych`'s `describe` and `describeBy` functions, you can also get values of summary statistics through using `dplyr`. While `psych::describeBy` can group along more than one attribute at time, this does not always work out well. The `summarize` function of `dplyr` does require you to specify which things you want but allows you to specify groups with the `group_by` function.

To demonstrate these, I'm going to create a table of summary statistics for the penguin data, grouping by species and sex.

```
# Code for Table 6
penguins %>%
  dplyr::select(species, sex, flipper_length_mm) %>%
  dplyr::group_by(species, sex) %>%
  tidyr::drop_na(everything()) %>%
  dplyr::summarize(
    n = n(),
    Min = min(flipper_length_mm),
    Q1 = quantile(flipper_length_mm, probs = 0.25),
    Median = median(flipper_length_mm),
    Q3 = quantile(flipper_length_mm, probs = 0.75),
    Max = max(flipper_length_mm),
    MAD = mad(flipper_length_mm),
    SAM = mean(flipper_length_mm),
    SASD = sd(flipper_length_mm),
    "Sample Skew" = psych::skew(flipper_length_mm),
    "Sample Ex. Kurtosis" = psych::kurtosi(flipper_length_mm),
    .groups = "keep"
  ) %>%
  dplyr::rename(
    Species = species,
    Sex = sex
  ) %>%
  knitr::kable(
    caption = "Summary Statistics Penguin Flipper Lengths by Species and Sex",
    digits = 3,
    format.args = list(big.mark = ","),
    align = c('l', 'l', rep('c', 11)),
    row.names = FALSE,
    format = "latex",
    booktabs = TRUE
  )  %>%
  kableExtra::kable_styling(
    font_size = 12,
```

```
    latex_options = c("scale_down", "HOLD_position", "striped")
  )
```

Table 6: Summary Statistics Penguin Flipper Lengths by Species and Sex

Species	Sex	n	Min	Q1	Median	Q3	Max	MAD	SAM	SASD	Sample Skew	Sample Ex. Kurtosis
Adelie	female	73	172	185.00	188.0	191.00	202	4.448	187.795	5.595	-0.292	0.393
Adelie	male	73	178	189.00	193.0	197.00	210	5.930	192.411	6.599	0.039	-0.070
Chinstrap	female	34	178	187.25	192.0	195.75	202	5.930	191.735	5.754	-0.391	-0.475
Chinstrap	male	34	187	196.00	200.5	203.00	212	6.672	199.912	5.977	0.172	-0.645
Gentoo	female	58	203	210.00	212.0	215.00	222	4.448	212.707	3.898	0.212	-0.469
Gentoo	male	61	208	218.00	221.0	225.00	231	5.930	221.541	5.673	-0.105	-0.725

Table 6 provides an alternative approach to using `psych::describeBy`. If you wanted you could store the output of the data cleaning calls (up to the `knitr::kable` call) as a data frame. Notice that in the data cleaning section, we not only got rid of `NA`s (`tidyr::drop_na(everything())`–part of `tidyverse`), we set the column names to what we wanted in `dplyr::summarize`, and we capitalized the names for species and sex `dplyr::rename`. This let us omit the `col.names` argument of `kable`. We also turned off the row names with `row.names = FALSE` in `kable`, as they aren't necessary here. The only other major alteration was adding two `'l'`s to the `align` argument for our two grouping attributes.

Interpreting Values

Interpreting values of statistics is important. First, doing so forces you to critically think about 1) what the statistic tells us about the data collection, and 2) how this value enriches the data story. Second, providing interpretations helps your readers to make connections and build their own understanding of the data and what *you* are trying to say about the data and the context. Finally, providing interpretations will better enable you to make decisions and defend those decisions.

I'm going to give example interpretative statements drawn from the oreo data context and looking at the mass (g) of créme filling in oreos.

- Size: There are 60 cookies in the data collection.
- *Sample Minimum*: The smallest amout of créme filling observed was 2.29 grams.
- *First Quartile*: One quarter (25% or ~15 cookies) had less than 3.25 grams of créme filling.
- *Sample Median*: One half (50% or 30 cookies) had less than 4.57 grams of créme filling.
- *Third Quartile*: One quarter (25% or ~15 cookies) had at least 6.41 grams of créme filling.
  - Notice that you can go either way for the quantiles.
- *Sample Maximum*: The most créme filling we observed as 7.21 grams.
- *Sample Range*: The smallest interval which contains all of our observations is 4.92 grams wide.
- *IQR*: The smallest interval which contains the middle 50% of our observations is 3.16 grams wide.
- *MAD*: Half of all possible pairs of cookies are less than 2.17 grams different (in absolute value).
- *SAM*: This collection of cookies amassed 4.83 times as much créme filling as cookies.
- *SASD*: The typical deviation in créme filling mass between pairs of cookies in our collection was approximately 1.66 grams.
- *Sample Skewness*: The positive value of *Sample Skewness* (i.e., 0.03) indicates that the underlying process will generate some cookies with *more* créme filling than most cookies.
  - Negative values would indicate *less* créme filling
  - A zero value indicates that process produces essentially just as many heavy and light cookies (symmetric)
  - **Important**: the numerical value is not as interpretable as you might think. For example a *sample kurtosis* of 8 does not produce cookies that are 8 times heavier than most cookies.
- *Sample Ex. Kurtosis*: The negative value of *Sample Ex. Kurtosis* indicates that the underlying process generates fewer outlier cookies than we would anticipate if a Guassian (normal) distribution described the process.
  - Positive values indicate the process generates *more* outlier cookies
  - A zero value inicates that the process generates just as many outlier cookies as we would expect of a Guassian (normal) distribution.
  - **Important**: don't attempt to over interpret the values. A value of 8 for *Sample Ex. Kurtosis* does not mean that the process produces 8 times as many outliers.

Final Remarks

This guide served two purposes: demonstrate how you can use R to calculate the values of various descriptive/incisive statistics and how to prepare tables of values in a document. Again, the best way to improve your skills is by practicing. Beyond the class demo Oreo data and the Palmer Penguins data, you can use the oreo data set assigned in HW #1.1, as well as the following data sets:

- Motor Trend Car Road Tests—access with `data(mtcars)`
- Weight versus Age of Chicks on Different Diets—access with `data(ChickWeight)`
- Effectiveness of Insect Sprays—access with `data(InsectSprays)`
- The Iris data set—access with `data(iris)`
- Palmer Penguin Data—install the `palmerpenguins` package first, then access with `palmerpenguins::penguins`

This concludes this guide to descriptive statistics in R/RStudio.