Lab 2: Descriptive Statistics

Describing comic sensibility is near impossible. It's sort of an abstract silliness, that sometimes the joke isn't the star. —Dana Carvey

The purpose of this lab is to show you how to compute basic descriptive statistics, including measures of central tendency (mean, mode, median) and variation (range, variance, standard deviation).

General Goals

1. Compute measures of central tendency using software
2. Compute measures of variation using software
3. Ask some questions of a data set using descriptive statistics

Important info

We will be using data from the gapminder project. You can download a small snippet of the data in .csv format from this link (note this dataset was copied from the gapminder library for R) gapminder.csv. If you are using R, then you can install the gapminder package. This method is described later in the R section.

R

Descriptives basics in R

We learned in lecture and from the textbook that data we want to use ask and answer questions often comes with loads of numbers. Too many numbers to look at all at once. That's one reason we use descriptive statistics. To reduce the big set of numbers to one or two summary numbers that tell use something about all of the numbers. R can produce descriptive statistics for you in many ways. There are base functions for most of the ones that you want. We'll go over some R basics for descriptive statistics, and then use our new found skills to ask some questions about real data.

Making numbers in R In order to do descriptive statistics we need to put some numbers in a variable. You can also do this using the `c()` command, which stands for combine

```
my_numbers <- c(1,2,3,4)
```

There a few other handy ways to make numbers. We can use `seq()` to make a sequence. Here's making the numbers from 1 to 100

```
one_to_one_hundred <- seq(1,100,1)
```

We can repeat things, using rep. Here's making 10 5s, and 25 1s:

```
rep(10,5)
```

```
## [1] 10 10 10 10 10
```

```
rep(1,25)
```

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

```
all_together_now <- c(rep(10,5),rep(1,25))
```

Sum Let's play with the number 1 to 100. First, let's use the `sum()` function to add them up

```r
one_to_one_hundred <- seq(1,100,1)
sum(one_to_one_hundred)
```

```
## [1] 5050
```

Length We put 100 numbers into the variable `one_to_one_hundred`. We know how many numbers there are in there. How can we get R to tell us? We use `length()` for that.

```r
length(one_to_one_hundred)
```

```
## [1] 100
```

Central Tendency

Mean Remember the mean of some numbers is their sum, divided by the number of numbers. We can compute the mean like this:

```r
sum(one_to_one_hundred)/length(one_to_one_hundred)
```

```
## [1] 50.5
```

Or, we could just use the `mean()` function like this:

```r
mean(one_to_one_hundred)
```

```
## [1] 50.5
```

Median The median is the number in the exact middle of the numbers ordered from smallest to largest. If there are an even number of numbers (no number in the middle), then we take the number in between the two (decimal .5). Use the `median` function. There's only 3 numbers here. The middle one is 2, that should be the median

```r
median(c(1,2,3))
```

```
## [1] 2
```

Mode R does not a base function for the Mode. You would have to write one for yourself. Here is an example of writing your own mode function, and then using it. Note I searched how to do this on Google, and am using the mode defined by this answer on stack overflow

Remember, the mode is the most frequently occurring number in the set. Below 1 occurs the most, so the mode will be one.

Note: When using custom functions like `mode`, ensure that the function is in the R environment. You can verify this in RStudio's Environment panel. Once it's in the environment, it should function correctly whether you place it in the source section or elsewhere.

```r
my_mode <- function(x) {
  ux <- unique(x)
  ux[which.max(tabulate(match(x, ux)))]
}

my_mode(c(1,1,1,1,1,1,1,2,3,4))
```

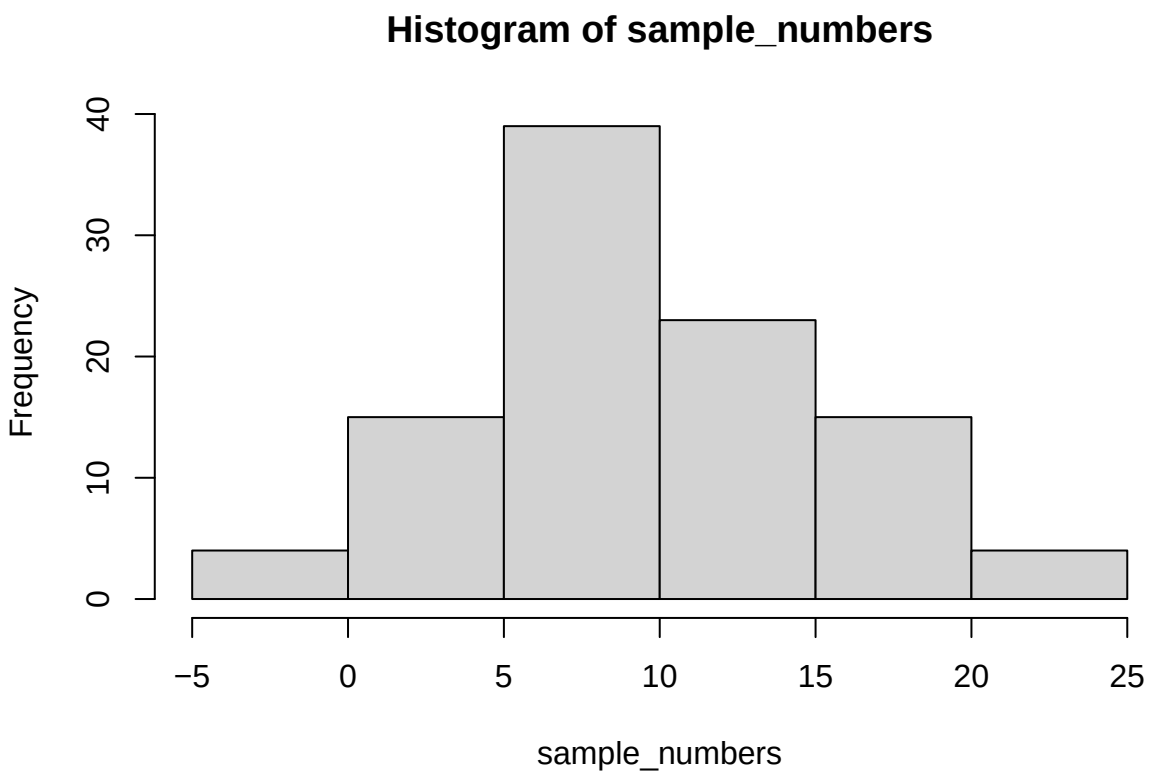
```
## [1] 1
```

Variation

We often want to know how variable the numbers are. We are going to look at descriptive statistics to describe this such as the **range**, **variance**, the **standard deviation**, and a few others.

First, let's remind ourselves what variation looks like (it's when the numbers are different). We will sample 100 numbers from a normal distribution (don't worry about this yet), with a mean of 10, and a standard deviation of 5, and then make a histogram so we can see the variation around 10..

```
sample_numbers <- rnorm(100,10,5)
hist(sample_numbers)
```



Note: The `rnorm()` function generates random numbers and can result in different outputs for each student. To keep the exercise consistent, you may use the `set.seed()` function before calling `rnorm()`. This sets a static seed for random number generation, ensuring that everyone gets the same random numbers for the exercise.

range The range is the minimum and maximum values in the set, we use the `range` function.

```
range(sample_numbers)
```

```
## [1] -2.086825 22.422328
```

var = variance We can find the sample variance using `var`. Note, divides by (n-1)

```
var(sample_numbers)
```

```
## [1] 29.49227
```

sd = standard deviation We find the sample standard deviation us SD. Note, divides by (n-1)

```
sd(sample_numbers)
```

```
## [1] 5.430679
```

Remember that the standard deviation is just the square root of the variance, see:

```
sqrt(var(sample_numbers))
```

```
## [1] 5.430679
```

All Descriptives Let's put all of the descriptives and other functions so far in one place:

```
sample_numbers <- rnorm(100,10,5)

sum(sample_numbers)
```

```
## [1] 1020.655
```

```
length(sample_numbers)
```

```
## [1] 100
```

```
mean(sample_numbers)
```

```
## [1] 10.20655
```

```
median(sample_numbers)
```

```
## [1] 10.40316
```

```
my_mode(sample_numbers)
```

```
## [1] 7.064998
```

```
range(sample_numbers)
```

```
## [1] -1.833618 21.726835
```

```
var(sample_numbers)
```

```
## [1] 23.14351
```

```
sd(sample_numbers)
```

```
## [1] 4.81077
```

Descriptives by conditions

Sometimes you will have a single variable with some numbers, and you can use the above functions to find the descriptives for that variable. Other times (most often in this course), you will have a big data frame of numbers, with different numbers in different conditions. You will want to find descriptive statistics for each the sets of numbers inside each of the conditions. Fortunately, this is where R really shines, it does it all for you in one go.

Let's illustrate the problem. Here I make a date frame with 10 numbers in each condition. There are 10 conditions, each labelled, A, B, C, D, E, F, G, H, I, J.

```
scores <- rnorm(100,10,5)
conditions <- rep(c("A","B","C","D","E","F","G","H","I","J"), each =10)
my_df <- data.frame(conditions,scores)
```

If you look at the `my_df` data frame, you will see it has 100 rows, there are 10 rows for each condition with a label in the `conditions` column, and 10 scores for each condition in the `scores` column. What if you wanted to know the mean of the scores in each condition? You would want to find 10 means.

The slow way to do it would be like this:

```
mean(my_df[my_df$conditions=="A",]$scores)
```

```
## [1] 10.51458
```

```
mean(my_df[my_df$conditions=="B",]$scores)
```

```
## [1] 9.689432
```

```
mean(my_df[my_df$conditions=="C",]$scores)
```

```
## [1] 8.844892
```

```
# and then keep going
```

Nobody wants to do that! Not, me I stopped doing it that way, you should to.

group_by and summarise We can easily do everything all at once using the `group_by` and `summarise` function from the `dplyr` package. Just watch

```
library(dplyr)
```

```
##
## Attaching package: 'dplyr'
```

```
## The following objects are masked from 'package:stats':
##
##     filter, lag
```

```
## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

```
my_df %>%
  group_by(conditions) %>%
  summarise(means = mean(scores))
```

```
## # A tibble: 10 x 2
##    conditions means
##    <chr>      <dbl>
##  1 A          10.5
##  2 B           9.69
##  3 C           8.84
##  4 D          10.1
##  5 E          10.9
##  6 F           8.69
##  7 G          10.1
##  8 H          12.2
##  9 I          10.5
## 10 J          12.5
```

A couple things now. First, the print out of this was ugly. Let's fix that. we put the results of our code into a new variable, then we use `knitr::kable` to print it out nicely when we `knit` the document

```
summary_df <- my_df %>%
               group_by(conditions) %>%
               summarise(means = mean(scores))

knitr::kable(summary_df)
```

conditions	means
A	10.514579
B	9.689432
C	8.844891
D	10.101629
E	10.914611
F	8.685423
G	10.075702
H	12.202343
I	10.453596
J	12.465102

multiple descriptives The best thing about the `dplyr` method, is that we can add more than one function, and we'll get more than one summary returned, all in a nice format, let's add the standard deviation:

```
summary_df <- my_df %>%
               group_by(conditions) %>%
               summarise(means = mean(scores),
                         sds = sd(scores))

knitr::kable(summary_df)
```

conditions	means	sds
A	10.514579	4.213253
B	9.689432	4.647020
C	8.844891	3.258603
D	10.101629	5.044184
E	10.914611	5.932771
F	8.685423	5.955467
G	10.075702	4.368820
H	12.202343	4.940175
I	10.453596	5.333311
J	12.465102	3.766398

We'll add the min and the max too:

```
summary_df <- my_df %>%
               group_by(conditions) %>%
               summarise(means = mean(scores),
                         sds = sd(scores),
                         min = min(scores),
                         max = max(scores))

knitr::kable(summary_df)
```

conditions	means	sds	min	max
A	10.514579	4.213253	1.6622480	15.46791
B	9.689432	4.647020	3.3262041	15.27472
C	8.844891	3.258603	4.7657550	15.12116
D	10.101629	5.044184	2.9375484	18.76588
E	10.914611	5.932771	1.7355520	19.95216

conditions	means	sds	min	max
F	8.685423	5.955467	2.8170466	20.78746
G	10.075702	4.368820	1.2842766	18.11670
H	12.202343	4.940175	5.0086791	19.38960
I	10.453596	5.333311	-0.3378016	19.29983
J	12.465102	3.766398	7.8710429	18.14517

Describing gapminder

Now that we know how to get descriptive statistics from R, we cam do this will some real data. Let's quickly ask a few question about the gapminder data.

```
library(gapminder)
gapminder_df <- gapminder
```

Note: The above code will only work if you have installed the gapminder package. Make sure you are connected to the internet, then choose the Packages tab from the bottom right panel, and choose install. Thens search for gapminder, choose it, and install it.

Another Note: Installing R packages should typically be done in the R console rather than in the RMarkdown document. This is a one-time action and doesn't need to be repeated each time you knit the document. To load an already installed package for use in your RMarkdown document, you can use the `library()` function within the setup chunk.

What are some descriptive for Life expectancy by continent? Copy the code from the last part of descriptives using `dplyr`, then change the names like this:

```
summary_df <- gapminder_df %>%
               group_by(continent) %>%
               summarise(means = mean(lifeExp),
                         sds = sd(lifeExp),
                         min = min(lifeExp),
                         max = max(lifeExp))

knitr::kable(summary_df)
```

continent	means	sds	min	max
Africa	48.86533	9.150210	23.599	76.442
Americas	64.65874	9.345088	37.579	80.653
Asia	60.06490	11.864532	28.801	82.603
Europe	71.90369	5.433178	43.585	81.757
Oceania	74.32621	3.795611	69.120	81.235

Generalization Exercise

(4 points - Pass/Fail)

Complete the generalization exercise described in your R Markdown document for this lab.

1. What is the mean, standard deviation, minimum and maximum life expectancy for all the gapminder data (across all the years and countries). Hint: do not use `group_by`
2. Calculate the mean, standard deviation, minimum, and maximum life expectancy across all continents for the year 2007, which is the most recent year available in the dataset. You can filter the data to this year by adding another pipe and using `filter(year == 2007) %>%`.

Writing assignment

(8 points - Graded)

Complete the writing assignment described in your R Markdown document for this lab. When you have finished everything. Knit the document and hand in your stuff (you can submit your .RMD file to blackboard if it does not knit.)

Your writing assignment is to answer these questions in full sentences using simple plain langauge:

1. Define the mode.
2. Explain what would need to happen in order for a set of numbers to have two modes
3. Define the median
4. Define the mean
5. Define the range
6. When calculating the standard deviation, explain what the difference scores represent
7. Explain why the difference scores are squared when calculating the standard deviation
8. If one set of numbers had a standard deviation of 5, and another had a standard deviation of 10, which set of numbers would have greater variance, explain why.

Rubric

General grading.

- You will receive 0 points for missing answers (say, if you do not answer question c, then you will receive 0 points for that question)
- You must write in complete sentences. Point form sentences will be given 0 points.
- Completely incorrect answers will receive 0 points.
- If your answer is generally correct but very difficult to understand and unclear you may receive half points for the question

Extra Practice Problems (Optional)

---

1. Using the life expectancy data set, produce a table of output showing the descriptive statistics (measures of central tendency and variability) for both years 1800 and 1934 (during the Great Depression).
2. Plot histograms of life expectancy for both years. How are these distributions different? (Hint: Plot these on the same axes so that they are comparable).