Lab 6 Descriptive Statistics

B. Sosnovski

03/06/2022

Descriptive Statistics

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

In this lab, you will learn the following:

1. To compute measures of central tendency using R
2. To compute measures of variation using R
3. To ask some questions about a data set using descriptive statistics

Important info

We will use the gapminder project data for this lab.

Making numbers in R

To do descriptive statistics, we 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)
my_numbers
```

```
## [1] 1 2 3 4
```

```
my_names <- c("Anna", "Bren", "Carlos", "Danny")
my_names
```

```
## [1] "Anna"   "Bren"   "Carlos" "Danny"
```

There are a few other handy ways to make numbers. We can use `seq()` to make a sequence.

```
one_to_forty <- seq(1,40,1)
one_to_forty
```

```
##  [1]  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25
## [26] 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40
```

```
even_numbers <- seq(2,39,2)
even_numbers
```

```
##  [1]  2  4  6  8 10 12 14 16 18 20 22 24 26 28 30 32 34 36 38
```

We can repeat things using `rep()`. Here's making five times the number 10 and eleven times the number 1:

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

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

```
rep(1,11)
```

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

```
all_together_now <- c(rep(10,5),rep(1,11))
all_together_now
```

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

Note: the syntax is `rep(object, number of repetitions)`

Example

Store the letter grades below in a variable called `grades`. A, B, B, B, B, A, B, C, B, C, C, C, A, C, B, B

```
Enter your code below:
```

Tabulating data

A common task when analyzing data is the construction of frequency tables. You can use several functions in R for that purpose, one of them being `table()`.

```
table(all_together_now)
```

```
## all_together_now
##  1 10
## 11  5
```

This shows the frequency of each value in the data.

Example

Use R to tabulate the letter grades from the previous example.

```
Enter your code below:
```

Sum

Let's play with the numbers from 1 to 100. To create this sequence of numbers, we use the function `seq()` and then use the `sum()` function to add them up.

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

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

Example

Use R to create the list of the numbers between 143 and 246 (integers), and calculate their sum. Use a variable called "my_numbers2".

```
Enter your code below:
```

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.

```
length(one_to_one_hundred)
```

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

Example

Find the length of the variable `my_numbers2` from the previous example.

```
Enter your code below:
```

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:

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

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

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

```
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 values (no number in the middle), then we take the number between the two (decimal .5). Use the `median()` function.

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

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

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

```
## [1] 2.5
```

Example

Use R to calculate the mean and the median of a list of numbers that contains twenty times the number 15, six times the number 8, and thirteen times the number 9.

```
Enter your code below:
```

Mode

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

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

```
Mode <- function(x) {
 a <- table(x)
 as.numeric(names(a)[a == max(a)])
}

x <- c(1,2,2,3,3,4)
Mode(x)
```

```
## [1] 2 3
```

Example

Use R to calculate the mode of 2,1,2,2,4,4,5,2

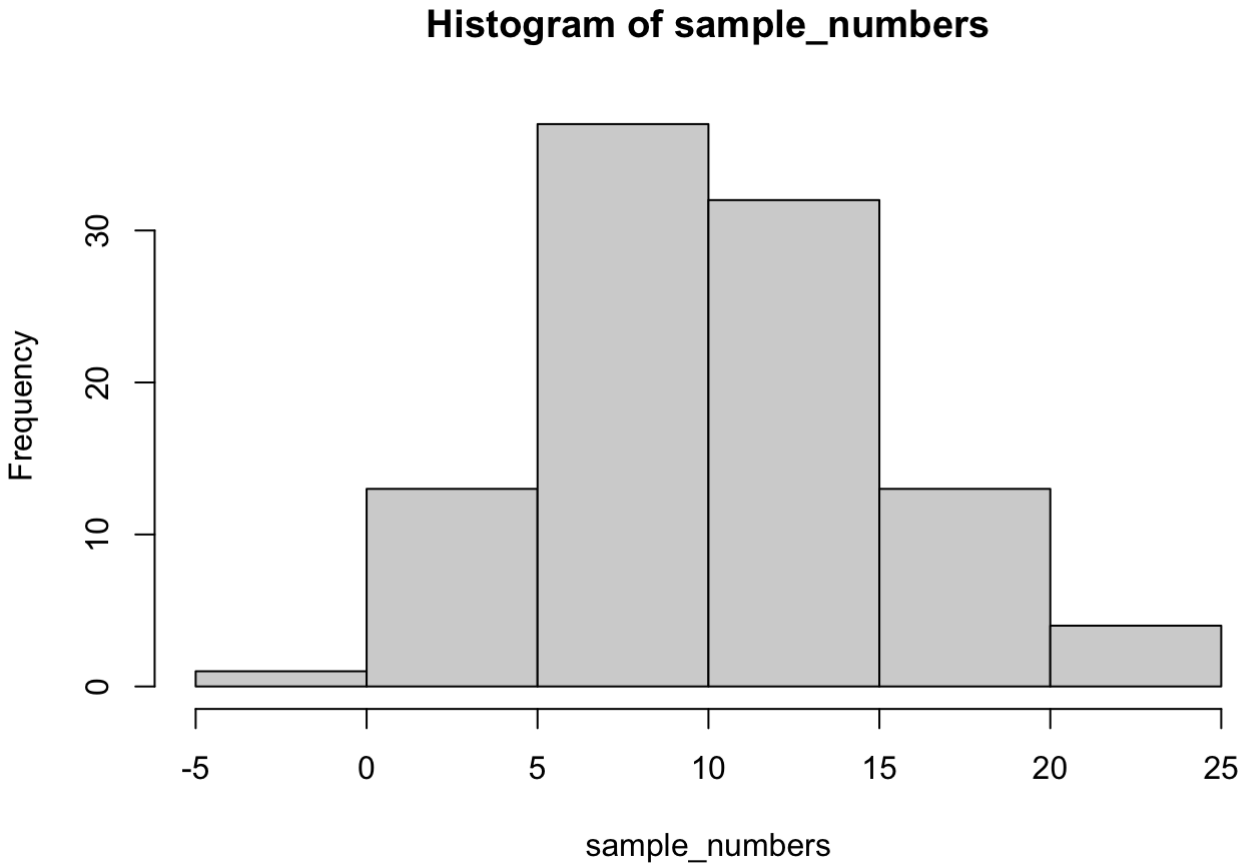
```
Enter your code below:
```

Variation

We often want to know how variable the numbers are. We will 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 (if we haven't covered this in the lecture yet, don't worry about it) with a mean of 10 and a standard deviation of 5 and then make a histogram to see the variation around 10.

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



range

The range in R is the *minimum* and *maximum* values in the set; we use the `range()` function [this is not the same as we computed in class].

```
range(sample_numbers)
```

```
## [1] -0.8023148 22.3451158
```

var = variance

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

```
var(sample_numbers)
```

```
## [1] 24.77019
```

sd = standard deviation

We find the sample standard deviation using `sd()`. Note: divides by (n-1)

```
sd(sample_numbers)
```

```
## [1] 4.976966
```

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

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

```
## [1] 4.976966
```

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] 972.1034
```

```
length(sample_numbers)
```

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

```
mean(sample_numbers)
```

```
## [1] 9.721034
```

```
median(sample_numbers)
```

```
## [1] 9.840207
```

```
Mode(sample_numbers)
```

```
##   [1] -5.876128 -3.557092 -1.610557 -1.364455  2.038547  2.446879  2.716807
##   [8]  2.857915  3.203215  3.414219  3.560047  3.680509  3.915945  4.010134
##  [15]  4.207583  4.259008  4.592616  4.810556  6.299655  6.313818  6.436958
##  [22]  6.528781  6.534385  6.581815  6.710463  6.835617  6.860096  6.934805
##  [29]  7.114593  7.181564  7.417618  7.533927  7.540664  7.698494  7.851187
##  [36]  7.862933  7.949608  7.951708  8.385656  8.619645  8.752677  8.864796
##  [43]  8.905462  9.065786  9.101403  9.106720  9.370581  9.667762  9.826138
##  [50]  9.832184  9.848229 10.044995 10.106572 10.128780 10.151572 10.317754
##  [57] 10.372619 11.028662 11.069949 11.288431 11.382409 11.435255 11.509361
##  [64] 11.655566 11.760190 11.786068 11.814417 11.943194 12.022185 12.125303
##  [71] 12.251735 12.377169 12.418439 12.542764 12.576148 12.938953 13.373730
##  [78] 13.393517 13.415657 13.745406 13.855387 14.017188 14.409490 14.516839
##  [85] 14.815887 14.957229 15.551579 15.642595 15.715567 15.945781 16.179648
##  [92] 16.570004 16.695359 16.908128 17.567889 18.250611 18.344952 18.956197
##  [99] 20.319324 23.115511
```

```
range(sample_numbers)
```

```
## [1] -5.876128 23.115511
```

```
var(sample_numbers)
```

```
## [1] 25.93281
```

```
sd(sample_numbers)
```

```
## [1] 5.092427
```

Example

Use R to find all descriptives of the variable `my_numbers2`.

```
Enter your code below:
```

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 (like categories). You will want to find descriptive statistics for each set of numbers inside each condition. Fortunately, this is where R 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 labeled A, B, C, D, E, F, G, H, I, J. We will generate normal distributions for the numbers.

```
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)
my_df
```

```
##     conditions     scores
## 1            A  1.9488356
## 2            A  7.6981907
## 3            A 15.1967531
## 4            A  5.2810534
## 5            A 10.2873946
## 6            A 12.9512402
## 7            A 15.3255847
## 8            A  9.7095527
## 9            A  5.8881785
## 10           A  4.6142500
## 11           B  0.6735651
## 12           B 15.3922816
## 13           B 15.0215193
## 14           B 15.8173150
## 15           B  6.0673248
## 16           B  6.0563548
## 17           B  3.4776414
## 18           B  5.2022458
## 19           B 10.8758230
## 20           B  3.5605096
## 21           C  9.0694576
## 22           C 13.8911559
## 23           C 12.0003001
## 24           C  7.3177749
## 25           C 13.6167187
## 26           C  8.5854365
## 27           C 10.0177840
## 28           C  0.2906766
## 29           C  1.7578970
## 30           C  9.9594927
## 31           D 14.7351400
## 32           D  6.7614749
## 33           D  8.4439133
## 34           D 11.7209461
## 35           D 18.8355190
## 36           D 14.3774039
## 37           D  9.3426156
## 38           D 13.1055677
## 39           D 15.4358847
## 40           D 11.5136795
## 41           E  7.5431744
## 42           E  7.6459553
## 43           E 16.7578872
## 44           E 11.0546467
## 45           E  5.3828260
## 46           E  9.7528000
## 47           E  7.7449872
## 48           E  2.5349572
## 49           E 15.4350087
## 50           E  2.5950722
## 51           F  8.8688935
```

```
## 52           F  8.4733282
## 53           F  4.9192875
## 54           F 10.0617912
## 55           F  6.6109064
## 56           F  2.9170453
## 57           F 13.6016567
## 58           F  3.1694675
## 59           F  7.5155358
## 60           F 14.9516434
## 61           G 11.9831329
## 62           G  8.3750147
## 63           G 14.2390604
## 64           G 21.8280457
## 65           G 10.1204978
## 66           G 13.7089846
## 67           G 15.9991271
## 68           G 17.0954494
## 69           G  6.7744162
## 70           G  7.5092844
## 71           H 11.7537669
## 72           H 13.1108306
## 73           H  8.6129383
## 74           H  5.7875353
## 75           H  6.4207179
## 76           H 10.5233025
## 77           H  7.9591960
## 78           H 16.4457328
## 79           H 11.3249000
## 80           H 15.4430849
## 81           I 20.8165536
## 82           I 18.9330352
## 83           I 16.5555362
## 84           I 20.5136299
## 85           I  6.2706811
## 86           I  7.3330056
## 87           I  3.4291746
## 88           I 13.1598561
## 89           I 11.2340328
## 90           I  6.7070449
## 91           J  6.3224070
## 92           J  1.4869977
## 93           J 11.1531062
## 94           J 13.9827597
## 95           J 12.9382041
## 96           J 12.9607289
## 97           J -2.8484868
## 98           J  1.0457738
## 99           J  8.5069350
## 100          J 17.4951273
```

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 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] 8.890103
```

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

```
## [1] 8.214458
```

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

```
## [1] 8.650669
```

```
# and then keep going
```

Nobody wants to do that!

group_by and summarize

We can do everything all at once using the `group_by` and `summarize` functions from the `dplyr` package.

```
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) %>%
  summarize(means = mean(scores))
```

```
## # A tibble: 10 × 2
##    conditions means
##    <chr>      <dbl>
##  1 A           8.89
##  2 B           8.21
##  3 C           8.65
##  4 D          12.4
##  5 E           8.64
##  6 F           8.11
##  7 G          12.8
##  8 H          10.7
##  9 I          12.5
## 10 J           8.30
```

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

```
summary_df <- my_df %>%
               group_by(conditions) %>%
               summarize(means = mean(scores))
knitr::kable(summary_df)
```

conditions	means
A	8.890103
B	8.214458
C	8.650669
D	12.427214
E	8.644732
F	8.108956
G	12.763301
H	10.738200
I	12.495255
J	8.304355

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	8.890103	4.597434
B	8.214458	5.598078
C	8.650669	4.549511
D	12.427214	3.628954
E	8.644732	4.790673
F	8.108956	4.022723
G	12.763301	4.770295
H	10.738200	3.607898
I	12.495255	6.451468
J	8.304355	6.629084

We'll add the min and the max too:

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

conditions	means	sds	min	max
A	8.890103	4.597434	1.9488356	15.32558
B	8.214458	5.598078	0.6735651	15.81732
C	8.650669	4.549511	0.2906766	13.89116
D	12.427214	3.628954	6.7614749	18.83552
E	8.644732	4.790673	2.5349572	16.75789
F	8.108956	4.022723	2.9170453	14.95164
G	12.763301	4.770295	6.7744162	21.82805
H	10.738200	3.607898	5.7875353	16.44573
I	12.495255	6.451468	3.4291746	20.81655
J	8.304355	6.629084	-2.8484868	17.49513

Example

1. Create a data frame with 5 conditions for every 20 numbers with normal distribution with a mean of 12 and a standard deviation of 3.
2. Calculate the mean and standard deviation for each condition.

```
Enter your code below:
```

Describing gapminder

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

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

```
## # A tibble: 1,704 × 6
##    country     continent  year lifeExp      pop gdpPercap
##    <fct>       <fct>     <int>   <dbl>    <int>     <dbl>
##  1 Afghanistan Asia       1952    28.8  8425333      779.
##  2 Afghanistan Asia       1957    30.3  9240934      821.
##  3 Afghanistan Asia       1962    32.0 10267083      853.
##  4 Afghanistan Asia       1967    34.0 11537966      836.
##  5 Afghanistan Asia       1972    36.1 13079460      740.
##  6 Afghanistan Asia       1977    38.4 14880372      786.
##  7 Afghanistan Asia       1982    39.9 12881816      978.
##  8 Afghanistan Asia       1987    40.8 13867957      852.
##  9 Afghanistan Asia       1992    41.7 16317921      649.
## 10 Afghanistan Asia       1997    41.8 22227415      635.
## # … with 1,694 more rows
```

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) %>%
               summarize(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

continent	means	sds	min	max
Europe	71.90369	5.433178	43.585	81.757
Oceania	74.32621	3.795611	69.120	81.235

Example

Calculate the mean, standard deviation, minimum and maximum for each continent's GDP per Capita.

```
Enter your code below:
```

References

The material used in this document contains excerpts and modifications from:

- Matthew J. C. Crump, Anjali Krishnan, Stephen Volz, and Alla Chavarga (2018) "Answering questions with data: Lab Manual." Last compiled on 2019-04-06. https://www.crumplab.com/statisticsLab/ (https://www.crumplab.com/statisticsLab/)