\

Descriptive Statistics

Biostatistics for Beginners

-

basic statistics for biologists

Erik Kusch
PhD Student
Aarhus University
Department of Bioscience
Section for Ecoinformatics & Biodiversity
Center for Biodiversity and Dynamics in a Changing World (BIOCHANGE)
Ny Munkegade 116, Building 1540
8000 Aarhus
Denmark
email: erik@i-solution.de

Summary:

These are the solutions to the exercises contained within the handout to "Descriptive Statistics" which walks you through the basics of descriptive statistics and its parameters. The analyses presented here are using data from the `StarWars` data set supplied through the `dplyr` package that have been saved as a .csv file.

Keep in mind that there is probably a myriad of other ways to reach the same conclusions as presented in these solutions.

Contents

1 Packages **2**
- 1.1 Basic Preamble . . . 2
- 1.2 Advanced Preamble . . . 2

2 Loading the Excel data into `R` **3**

3 What's contained within our data? **3**
- 3.1 Dimensions . . . 3
- 3.2 Modes . . . 4
- 3.3 Data Content . . . 4

4 Parameters of descriptive statistics **5**
- 4.1 Names . . . 5
- 4.2 Height . . . 5
  - 4.2.1 Subsetting . . . 5
  - 4.2.2 Location Parameters . . . 5
  - 4.2.3 Distribution Parameters . . . 6
  - 4.2.4 Summary . . . 6
- 4.3 Mass . . . 6
  - 4.3.1 Subsetting . . . 6
  - 4.3.2 Location Parameters . . . 6
  - 4.3.3 Distribution Parameters . . . 7
- 4.4 Hair Color . . . 7
- 4.5 Eye Colour . . . 7
- 4.6 Birth Year . . . 7
  - 4.6.1 Subsetting . . . 7
  - 4.6.2 Location Parameters . . . 8
  - 4.6.3 Distribution Parameters . . . 8
- 4.7 Gender . . . 8
- 4.8 Closing . . . 8

1. Packages

As you will remember from our lecture slides, the calculation of the mode in R can either be achieved through some intense coding or simply by using the `mlv(..., method="mfv")` function contained within the `modeest` package (unfortunately, this package is out of date and can sometimes be cahellenging to install).

Conclusively, it is now time for you to get familiar with how packages work in R. Packages are the way by which R is supplied with user-created and moderator-mediated functionality that exceeds the base applicability of R. Many things you will want to accomplish in more advanced statistics is impossible without such packages and even basic tasks such as data visualisation (dealt with in our next seminar) are reliant on R packages.

If you want to get a package and its functions into R there are two ways we will discuss in the following. In general, it pays to load all packages at the beginning of a coding document before any actual analyses happen (in the preamble) so you get a good overview of what the program is calling upon.

1.1 Basic Preamble

This is the most basic version of getting packages into R and is widely practised and taught. Unsurprisingly, I am not a big fan of it.

First, you use function `install.packages()` to download the desired package off dedicated servers (usually CRAN-mirrors) to your machine where it is then unpacked into a library (a folder that's located in your documents section by default). Secondly, you need to invoke the `library()` function to load the R package you need into your active R session. In our case of the package `modeest` it would look something like this:

```
install.packages("modeest")
library(modeest)
```

The reason I am not overly fond of this procedure is that it is clunky, can break easily through spelling mistakes and starts cluttering your preamble super fast if the analyses you are wanting to perform require excessive amounts of packages. Additionally, when you are some place with a bad internet connection you might not want to re-download packages that are already contained on your hard drive.

1.2 Advanced Preamble

There is a myriad of different preamble styles (just as there are tons of different, personalised coding styles). I am left with presenting my preamble of choice here but I do not claim that this is the most sophisticated one out there.

The way this preamble works is that it is structured around a user-defined function (something we will touch on later in our seminar series) which first checks whether a package is already downloaded and then installs (if necessary) and/or loads it into R. This function is called `install.load.package()` and you can see its specification down below (don't worry if it doesn't make sense to you yet - it is not supposed to at this point). Unfortunately, it can only ever be applied to one package at a time and so we need a workaround to make it work on multiple packages at once. This can be achieved by establishing a vector of all desired package names (`package_vec`) and then applying (`sapply()`) the `install.load.package()` function to every item of the package name vector iteratively as follows:

```
# function to load packages and install them if they haven't been installed yet
install.load.package <- function(x) {
    if (!require(x, character.only = TRUE))
        install.packages(x)
    require(x, character.only = TRUE)
}
# packages to load/install if necessary
```

```r
package_vec <- c("modeest")
# applying function install.load.package to all packages specified in package_vec
sapply(package_vec, install.load.package)
```

```
## modeest
##    TRUE
```

Why do I prefer this? Firstly, it is way shorter than the basic method when dealing with many packages (which you will get into fast, I promise), reduces the chance for typos by 50% and does not override already installed packages hence speeding up your processing time.

2. Loading the Excel data into R

Our data is located in the `Data` folder and is called `DescriptiveData.csv`. Since it is a .csv file, we can simply use the `R` in-built function `read.csv()` to load the data by combining the former two identifiers into one long string with a backslash separating the two (the backslash indicates a step down in the folder hierarchy). Given this argument, `read.csv()` will produce an object of type `data.frame` in `R` which we want to keep in our environment and hence need to assign a name to. In our case, let that name be `Data_df` (I recommend using endings to your data object names that indicate their type for easier coding without constant type checking):

```r
Data_df <- read.csv("Data/DescriptiveData.csv")  # load data file from Data folder
```

3. What's contained within our data?

Now that our data set is finally loaded into `R`, we can finally get to trying to make sense of it. Usually, this shouldn't ever be something one has to do in `R` but should be manageable through a project-/data-specific README file (we will cover this in our seminar on hypotheses testing and project planning) but for now we are stuck with pure exploration of our data set. Get your goggles on and let's dive in!

Firstly, it always pays to asses the basic attributes of any data object (remember the Introduction to `R` seminar):

- *Name* - we know the name (it is `Data_df`) since we named it that
- *Type* - we already know that it is a `data.frame` because we created it using the `read.csv` function
- *Mode* - this is an interesting one as it means having to subset our data frame
- *Dimensions* - a crucial information about how many observations and variables are contained within our data set

3.1 Dimensions

Let's start with the *dimensions* because these will tell us how many *modes* (these are object attribute modes and not descriptive parameter modes) to asses:

```r
dim(Data_df)
```

```
## [1] 87  8
```

Using the `dim()` function, we arrive at the conclusion that our `Data_df` contains 87 rows and 8 columns. Since data frames are usually ordered as observations $\times$ variables, we can conclude that we have 87 observations and 8 variables at our hands.
You can arrive at the same point by using the function `View()` in `R`. I'm not showing this here because it does not translate well to paper and would make whoever chooses to print this waste paper.

3.2 Modes

Now it's time to get a hang of the *modes* of the variable records within our data set. To do so, we have two choices, we can either subset the data frame by columns and apply the `class()` function to each column subset or simply apply the `str()` function to the data frame object. The reason `str()` may be favourable in this case is due to the fact that `str()` automatically breaks down the structure of `R`-internal objects and hence saves us the sub-setting:

```
str(Data_df)
```

```
## 'data.frame':    87 obs. of  8 variables:
##  $ name      : Factor w/ 87 levels "Ackbar","Adi Gallia",..: 46 15 62 21 44 54 10 64 12 53 ...
##  $ height    : int  172 167 96 202 150 178 165 97 183 182 ...
##  $ mass      : num  77 75 32 136 49 120 75 32 84 77 ...
##  $ hair_color: Factor w/ 13 levels "","auburn","auburn, grey",..: 6 1 1 11 8 9 8 1 5 4 ...
##  $ skin_color: Factor w/ 31 levels "blue","blue, grey",..: 7 9 29 28 17 17 17 30 17 7 ...
##  $ eye_color : Factor w/ 15 levels "black","blue",..: 2 15 11 15 4 2 2 11 4 3 ...
##  $ birth_year: num  19 112 33 41.9 19 52 47 NA 24 57 ...
##  $ gender    : Factor w/ 5 levels "","female","hermaphrodite",..: 4 1 1 4 2 4 2 1 4 4 ...
```

As it turns out, our data frame knows the 8 variables of name, height, mass, hair_color, skin_color, eye_color, birth_year, gender which range from `integer` to `numeric` and `factor` modes.

3.3 Data Content

So what does our data actually tell us? Answering this question usually comes down to some analyses but for now we are only really interested in what kind of information our data frame is storing.

Again, this would be easiest to asses using a README file or the `View()` function in `R`. However, for the sake of brevity we can make due with the following to commands which present the user with the first and last five rows of any respective data frame:

```
head(Data_df)
```

```
##              name height mass  hair_color  skin_color eye_color birth_year gender
## 1 Luke Skywalker    172   77       blond        fair      blue         19   male
## 2          C-3PO    167   75                    gold    yellow        112       
## 3          R2-D2     96   32             white, blue       red         33       
## 4    Darth Vader    202  136        none       white    yellow         42   male
## 5    Leia Organa    150   49       brown       light     brown         19 female
## 6      Owen Lars    178  120 brown, grey       light      blue         52   male
```

```
tail(Data_df)
```

```
##              name height mass hair_color skin_color eye_color birth_year gender
## 82           Finn     NA   NA      black       dark      dark         NA   male
## 83            Rey     NA   NA      brown      light     hazel         NA female
## 84    Poe Dameron     NA   NA      brown      light     brown         NA   male
## 85            BB8     NA   NA       none       none     black         NA   none
## 86 Captain Phasma     NA   NA    unknown    unknown   unknown         NA female
## 87  Padmé Amidala    165   45      brown      light     brown         46 female
```

The avid reader will surely have picked up on the fact that all the records in the `name` column of `Data_df` belong to characters from the Star Wars universe. In fact, this data set is a modified version of the `StarWars` data set supplied by the `dplyr` package and contains information of many of the important cast members of the Star Wars movie universe.

4. Parameters of descriptive statistics

4.1 Names

As it turns out (and should've been obvious from the onset if we're honest), every major character in the cinematic Star Wars Universe has a unique name to themselves. Conclusively, the calculation of any parameters of descriptive statistics makes no sense with the names of our characters for the two following reasons:

- The name variable is of mode character/factor and only allows for the calculation of the mode
- Since every name only appears once, there is no mode

As long as the calculation of descriptive parameters of the `name` variable of our data set is concerned, Admiral Ackbar said it best.

4.2 Height

Let's get started on figuring out some parameters of descriptive statistics for the `height` variable of our Star Wars characters.

4.2.1 Subsetting

First, we need to extract the data in question from our big data frame object. This can be achieved by indexing through the column name as follows:

```
Height <- Data_df$height
```

4.2.2 Location Parameters

Now, with our `Height` vector being the numeric height records of the Star Wars characters in our data set, we are primed to calculate location parameters as follows:

```
mean <- mean(Height, na.rm = TRUE)
median <- median(Height, na.rm = TRUE)
mode <- mlv(na.omit(Height), method = "mfv")
min <- min(Height, na.rm = TRUE)
max <- max(Height, na.rm = TRUE)
range <- max - min

# Combining all location parameters into one vector for easier viewing
LocPars_vec <- c(mean, median, mode, min, max, range)
names(LocPars_vec) <- c("mean", "median", "mode", "minimum", "maximum", "range")
LocPars_vec
```

```
##    mean  median    mode minimum maximum   range
##     174     180     183      66     264     198
```

As you can clearly see, there is a big range in the heights of our respective Star Wars characters with mean and median being fairly disjunct due to the outliers in height on especially either end.

4.2.3 Distribution Parameters

Now that we are aware of the location parameters of the Star Wars height records, we can move on to the distribution parameters/parameters of spread. Those can be calculated in R as follows:

```
var <- var(Height, na.rm = TRUE)
sd <- sd(Height, na.rm = TRUE)
quantiles <- quantile(Height, na.rm = TRUE)

# Combining all location parameters into one vector for easier viewing
DisPars_vec <- c(var, sd, quantiles)
names(DisPars_vec) <- c("var", "sd", "0%", "25%", "50%", "75%", "100%")
DisPars_vec
```

```
##  var   sd   0%  25%  50%  75% 100%
## 1209   35   66  167  180  191  264
```

Notice how some of the quantiles (actually quartiles in this case) link up with some of the parameters of central tendency.

4.2.4 Summary

Just to round this off, have a look at what the `summary()` function in R supplies you with:

```
summary <- summary(na.omit(Height))
summary
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##      66     167     180     174     191     264
```

This is a nice assortment of location and dispersion parameters.

4.3 Mass

Now let's focus on the weight of our Star Wars characters.

4.3.1 Subsetting

Again, we need to extract our data from the data frame. For the sake of brevity, I will refrain from showing you the rest of the analysis and only present its results to save some space.

```
Mass <- Data_df$mass
```

4.3.2 Location Parameters

```
##    mean  median    mode minimum maximum   range
##      97      79      80      15    1358    1343
```

As you can see, there is a huge range in weight records of Star Wars characters and especially the outlier on the upper end (1358kg) push the mean towards the upper end of the weight range and away from the median. We've got Jabba Desilijic Tiure to thank for that.

4.3.3 Distribution Parameters

```
##   var    sd    0%   25%   50%   75%  100%
## 28716   169    15    56    79    84  1358
```

Quite obviously, the wide range of weight records also prompts a large variance and standard deviation.

4.4 Hair Color

Hair colour in our data set is saved in column 4 of our data set and so when sub-setting the data frame to obtain information about a characters hair colour, instead of calling on `Data_df$hair_color` we can also do so as follows:

```
HCs <- Data_df[, 4]
```

Of course, hair colour is not a `numeric` variable and much better represent by being of mode `factor`. Therefore, we are unable to obtain most parameters of descriptive statistics but we can show a frequency count as follows which allows for the calculation of the mode:

```
table(HCs)
```

```
## HCs
##                      auburn  auburn, grey auburn, white         black         blond
##             5              1             1             1            13             3
##        blonde         brown   brown, grey          grey          none       unknown
##             1             18             1             1            37             1
##         white
##             4
```

4.5 Eye Colour

Eye colour is another `factor` mode variable:

```
ECs <- Data_df$eye_color
```

We can only calculate the mode by looking for the maximum in our `table()` output:

```
table(ECs)
```

```
## ECs
##         black          blue     blue-gray         brown          dark          gold
##            10            19             1            21             1             1
## green, yellow         hazel        orange          pink           red     red, blue
##             1             3             8             1             5             1
##       unknown         white        yellow
##             3             1            11
```

4.6 Birth Year

4.6.1 Subsetting

As another `numeric` variable, birth year allows for the calculation of the full range of parameters of descriptive statistics:

```
BY <- Data_df$birth_year
```

Keep in mind that StarWars operates on a different time reference scale than we do.

4.6.2 Location Parameters

```
##    mean  median    mode minimum maximum   range    <NA>    <NA>    <NA>    <NA>    <NA>
##      88      52      19      42      48      52      72      82      92       8     896
##    <NA>
##     888
```

Again, there is a big disparity here between mean and median which stems from extreme outliers on both ends of the age spectrum (Yoda and Wicket Systri Warrick, respectively).

4.6.3 Distribution Parameters

```
##   var    sd    0%   25%   50%   75%  100%
## 23929   155     8    35    52    72   896
```

Unsurprisingly, there is a big variance and standard deviation for the observed birth year/age records.

4.7 Gender

Gender is another `factor` mode variable (obviously):

```
Gender <- Data_df$gender
```

We can, again, only judge the mode of our data from the output of the `table()` function:

```
table(Gender)
```

```
## Gender
##                      female hermaphrodite          male          none
##             3             19             1            62             2
```

Keep in mind that StarWars characters can have some weird genders.

4.8 Closing

Congratulations on making it through this seminar. Descriptive statistics are a gateway into higher and more advanced statistic approaches so this feels appropriate:


WHEN YOU BOOST YOUR STATS
TO THE HIGHEST LEVEL:
I've become more powerful than
any Jedi has ever dreamed of