Lab 2 Tutorial

Janette Avelar & Anwesha Guha

1/25/2022

Intro

This week we will go over descriptive statistics.

Note: **All material is taken from Dr. Zopluoglu's 2021 EDUC 614 materials and adapted for this year's course by Anwesha for this week.**

We have adapted it to slide format using R. Want to learn how we created R slide presentations? This tutorial is helpful.

Importing Data

Importing Data

There are many ways of importing data into R depending on your taste and style. In class, you practiced how to import data using `Import Dataset` functionality under the Environment tab.

If you want to follow along, go ahead and import the *Add.csv* file into R following the same instructions.

Make sure you name the data object as *Add* because the rest of the code will assume your data object name in the Environment is Add.

Importing Data

In addition, I would also like to show you how you can import the same file with base functions using syntax. Below is how you can import the Add.csv file using the `read.csv` function. For more information, you can type `?read.csv` in the R console and read more about how to use this function.

```
Add <- read.csv(file   = "~/Downloads/Add.csv",
                header = TRUE)
```

Note that “Add <-” indicates that we are asking R to create a new object with a name Add using the syntax followed by assignment sign, “<-”.

Importing Data

There are many arguments `read.csv` function can take. Here, I use two main arguments. The first one is indicating that the location of the file in my computer.

file = "/Users/aguha/Downloads/Add.csv" provides the path for R to find the file I want to import.

The second argument is "header=TRUE".

This argument is a `logical` argument and can take only two values: TRUE or FALSE. In this case, we set header=TRUE' because the first line of the file has the column labels so R reads the first line as column names, then reads the rest of the lines as data. If you run this code in your console, you will see that it imports the file, and you will a new data object created in your Environment.

After Importing Data

You can always do some quick checks when you import a new dataset. You use str() function to check the internal structure of an R object.

```
str(Add)
```

```
## 'data.frame':    88 obs. of  8 variables:
##  $ CaseNum: int  1 2 3 4 5 6 7 8 9 10 ...
##  $ ADDSC  : int  45 50 49 55 39 68 69 56 58 48 ...
##  $ Sex    : int  1 1 1 1 1 1 1 1 1 1 ...
##  $ Repeat : int  0 0 0 0 0 1 1 0 0 0 ...
##  $ IQ     : int  111 102 108 109 118 79 88 102 105 92 ..
##  $ GPA    : num  2.6 2.75 4 2.25 3 1.67 2.25 3.4 1.33 3.
##  $ SocProb: int  0 0 0 0 0 0 1 0 0 0 ...
##  $ Dropout: int  0 0 0 0 0 1 1 0 0 0 ...
```

This indicates that "Add" is a data.frame object, has 88 observations (number of rows) and 8 variables (number of columns). It also provides a list of the variables in this data frame.

After Importing Data

You can also explicitly call the names of the columns in this dataset using colnames() function.

```
colnames(Add)
```

```
## [1] "CaseNum" "ADDSC"   "Sex"      "Repeat"  "IQ"       "(
## [8] "Dropout"
```

You can always run the following code, and it will open your data as a new tab next to your Source code. You can't edit or manipulate your data in this tab. You can only look at it.

```
View(Add)
```

Missing Data

Dealing with Missing Data

Suppose, that you have certain values for certain variables that represent missing data. In this dataset, we know that there are two variables with missingness: Repeat and Dropout

- 9 and 99 represent missingness for Sex
- 999 represents missingness for Dropout

So, I will recode the data such that these values for this particular columns are converted to NA, the internal code R recognizes missingness.

```
Add$Sex[Add$Sex==9] <- NA
Add$Sex[Add$Sex==99] <- NA

Add$Dropout[Add$Dropout==999] <- NA
```

The code above uses base functions, taking the value that you specify for the variable and replacing with a different value (NA in this case).

Categorical Variables

Categorical Variables

When you import a dataset, some categorical variables are labeled using strings, e.g. Yes/No, TRUE/FALSE, Asian/White/Hispanic/American Indian/...

Most of the times, however, you will see numeric codes for these variables, and R recognizes them as numbers, which is not appropriate.

For instance, in this dataset, Repeat is coded as 0 and 1, and 0 means did not repeat, while 1 means repeated. Or, SocProb is also coded as 0 and 1, and 0 means No while1 means Yes.

Categorical Variables

If you run str() on Add, you will see that all these variables are simply recognized as a numeric or integer variable:

```
str(Add)
```

```
## 'data.frame':    88 obs. of  8 variables:
##  $ CaseNum: int  1 2 3 4 5 6 7 8 9 10 ...
##  $ ADDSC  : int  45 50 49 55 39 68 69 56 58 48 ...
##  $ Sex    : int  1 1 1 1 1 1 1 1 1 1 ...
##  $ Repeat : int  0 0 0 0 0 1 1 0 0 0 ...
##  $ IQ     : int  111 102 108 109 118 79 88 102 105 92 ..
##  $ GPA    : num  2.6 2.75 4 2.25 3 1.67 2.25 3.4 1.33 3.
##  $ SocProb: int  0 0 0 0 0 0 1 0 0 0 ...
##  $ Dropout: int  0 0 0 0 0 1 1 0 0 0 ...
```

Categorical Variables

Convert these variables to Factor before you move forward with your analysis, as Factors are the accurate representation of categorical variables in R. We can convert any numeric variable to a Factor using the factor function.

We will create a new column by converting the numeric version of these variables to factors by creating and appending a new column to the dataset. I will add .f extension for the new variable names so we can distinguish them from the original. Let's start with Sex, coded as 1=male, 2=female.

```
str(Add$Sex)
```

```
##  int [1:88] 1 1 1 1 1 1 1 1 1 1 ...
```

```
table(Add$Sex)
```

```
##
##  1  2
## 51 31
```

Categorical Variables

Convert to factor.

```r
Add$Sex.f <- factor(x      = Add$Sex,
                    levels = c(1,2),
                    labels = c('Male','Female'))
table(Add$Sex.f)
```

```
##
##   Male Female
##     51     31
```

Categorical Variables

All 1s are now Male, and All 2s are now Female.

- levels = c(1,2), this indicates that the original numeric variable contains two values in the dataset 1 and 2
- labels = c('Male','Female'), this indicates the matching labels I want to use for 1 and 2. These labels are arbitrary. For instance, you can say labels = c('F','M'), and 1 and 2 will be represented by F and M labels, respectively. It is up to you if the data dictionary does not already indicate the corresponding values.

Categorical Variables

Also, if you run str(Add) now, you should notice the new variable you compute (Sex.f) is recognized as a Factor with 2 levels.

```
str(Add)
```

```
## 'data.frame':    88 obs. of  9 variables:
##  $ CaseNum: int  1 2 3 4 5 6 7 8 9 10 ...
##  $ ADDSC  : int  45 50 49 55 39 68 69 56 58 48 ...
##  $ Sex    : int  1 1 1 1 1 1 1 1 1 1 ...
##  $ Repeat : int  0 0 0 0 0 1 1 0 0 0 ...
##  $ IQ     : int  111 102 108 109 118 79 88 102 105 92 .
##  $ GPA    : num  2.6 2.75 4 2.25 3 1.67 2.25 3.4 1.33 3
##  $ SocProb: int  0 0 0 0 0 0 1 0 0 0 ...
##  $ Dropout: int  0 0 0 0 0 1 1 0 0 0 ...
##  $ Sex.f  : Factor w/ 2 levels "Male","Female": 1 1 1 1
```

Categorical Variables

Similarly, I will convert all categorical variables in the dataset by creating and appending a new variable at the end.

```r
Add$Repeat.f <- factor(x      = Add$Repeat,
                       levels = c(0,1),
                       labels = c('No','Yes'))

Add$SocProb.f <- factor(x      = Add$SocProb,
                        levels = c(0,1),
                        labels = c('No','Yes'))

Add$Dropout.f <- factor(x      = Add$Dropout,
                        levels = c(0,1),
                        labels = c('No','Yes'))
```

Categorical Variables

Then, check the dataset with the new variables.

```
str(Add)
```

```
## 'data.frame':    88 obs. of  12 variables:
##  $ CaseNum  : int  1 2 3 4 5 6 7 8 9 10 ...
##  $ ADDSC    : int  45 50 49 55 39 68 69 56 58 48 ...
##  $ Sex      : int  1 1 1 1 1 1 1 1 1 1 ...
##  $ Repeat   : int  0 0 0 0 0 1 1 0 0 0 ...
##  $ IQ       : int  111 102 108 109 118 79 88 102 105 92
##  $ GPA      : num  2.6 2.75 4 2.25 3 1.67 2.25 3.4 1.33
##  $ SocProb  : int  0 0 0 0 0 0 1 0 0 0 ...
##  $ Dropout  : int  0 0 0 0 0 1 1 0 0 0 ...
##  $ Sex.f    : Factor w/ 2 levels "Male","Female": 1 1 1
##  $ Repeat.f : Factor w/ 2 levels "No","Yes": 1 1 1 1 1 2
##  $ SocProb.f: Factor w/ 2 levels "No","Yes": 1 1 1 1 1 1
##  $ Dropout.f: Factor w/ 2 levels "No","Yes": 1 1 1 1 1 2
```

Indexing in a Data Frame

Indexing in a Data Frame

Sometimes, you just want to call certain elements from the data frame instead of looking at the whole data.

For instance let's say you want to see the value on Row 32 column 6 in your data set.

```
Add[32,6]
```

```
## [1] 2.25
```

This will return 2.25. It indicates that the value for the variable in the 6th column (which is GPA) for the individual on Row 32 is 2.25.

Indexing in a Data Frame

Notice how we index the data frame using double brackets

[i,j]

So, the index value before comma inside the double brackets (i) refers to the row number while the index value after comma inside the double brackets refers to the column number. See another example:

```r
Add[10,8]
```

```
## [1] 0
```

This should return the value in your dataset for the 10th individual (Row 10), on the 8th column variable(Dropout).

Indexing in a Data Frame

Sometimes you want to call all values in a row. If that's the case, then you can leave the space after comma inside double brackets empty.

For instance, the syntax below will print all the values on the third row in your console as a vector.

```
Add[3,]
```

```
##   CaseNum ADDSC Sex Repeat  IQ GPA SocProb Dropout Sex.f
## 3       3    49   1      0 108   4       0       0  Male
##   Dropout.f
## 3        No
```

Indexing in a Data Frame

Or, you can similarly call all values in a column. If that's the case, then you can leave the space before comma inside double brackets empty.

For instance, the syntax below will print all the values on the 6th column (GPA) in your console as a vector.

```
Add[,6]
```

```
##  [1] 2.60 2.75 4.00 2.25 3.00 1.67 2.25 3.40 1.33 3.50
## [16] 2.50 3.55 2.75 3.50 2.75 3.00 3.25 3.30 2.75 3.75
## [31] 0.67 2.25 1.75 3.25 1.75 3.00 3.00 3.00 3.25 2.25
## [46] 2.75 2.67 2.00 2.00 1.67 3.00 1.50 3.75 0.67 1.50
## [61] 3.75 2.67 3.50 0.67 2.25 3.00 3.50 1.75 1.00 3.00
## [76] 1.00 2.25 1.00 2.50 2.75 0.75 1.30 1.25 1.50 3.00
```

Indexing in a Data Frame

There are other ways you can call all the values from a columnn. First, you can use the column label (note: this is how we recoded the values to NA when we were dealing with missingness).

```
Add$GPA
```

There is another way of calling the same column. This is like a combined version of the two methods above.

```
Add[,'GPA']
```

Indexing in a Data Frame

Sometimes, you just want to see a certain segment of data. For instance, let's say you want to see the the data from Row 10 to Row 15, and Column 2 to Column 5.

```
Add[10:15, 2:5]
```

```
##    ADDSC Sex Repeat  IQ
## 10    48   1      0  92
## 11    34   1      0 131
## 12    50   2      0 104
## 13    85  NA      0  83
## 14    49   1      0  84
## 15    51   1      0  85
```

- 10:20 before comma indicates that we request Row 10 to Row 20
- 2:5 after comma indicates that we request Column 2 to Column 5

Indexing in a Data Frame

What if the rows and columns you would like to see are not consecutive? For instance, let's say you want to see Row 10, Row 35, and Row 80. Notice that c(10,35,80) before comma inside the brackets provides a list of row numbers I want. c() is special way in R to create a vector of numbers. There is no number specified after comma inside the brackets, so it means we request R to give all the columns in the data.

```
Add[c(10,35,80),]
```

```
##    CaseNum ADDSC Sex Repeat  IQ  GPA SocProb Dropout Sex
## 10      10    48   1      0  92 3.50       0       0  Ma
## 35      35    46   1      0 103 1.75       0       0  Ma
## 80      80    48   1      0  86 2.75       0       0  Ma
##    Dropout.f
## 10        No
## 35        No
## 80        No
```

Indexing in a Data Frame

We can do the same thing with columns. For instance, let's say you want to see Column 1, Column 2, and Column 6. Notice that c(1,2,6) after comma inside the brackets provides a list of column numbers I want. There is no number specified before comma inside the brackets, so it means we request R to give all the rows in the data.

```
Add[,c(1,2,6)]
```

```
##    CaseNum ADDSC  GPA
## 1        1    45 2.60
## 2        2    50 2.75
## 3        3    49 4.00
## 4        4    55 2.25
## 5        5    39 3.00
## 6        6    68 1.67
## 7        7    69 2.25
## 8        8    56 3.40
## 9        9    58 1.33
```

Indexing in a Data Frame

You can combine these two approaches. For instance, let's say you want to see Row 5, Row 10, Row 35, and Row 80 and Column 1, Column 2, and Column 6.

```
Add[c(5,10,35,80),c(1,2,6)]
```

```
##    CaseNum ADDSC  GPA
## 5        5    39 3.00
## 10      10    48 3.50
## 35      35    46 1.75
## 80      80    48 2.75
```

There is another way of doing the same thing.

```
Add[c(5,10,35,80),c('CaseNum','ADDSC','GPA')]
```

Instead of column numbers, I can just list the column names (with quotes) after comma inside the bracket.

Computing Descriptive Statistics

Computing Descriptive Statistics

There are many ways in R to calculate descriptive statistics. The one I most like is the describe() function from the psych package.

The code below will install the psych package into your computer. You have to do it only once.

```
install.packages('psych')
```

Next, you have to load the package to the R environment. You have to do it every time you close and reopen RStudio. You can use either function below.

```
require(psych) ##OR
library(psych)
```

Once you load the psych package into your Environment, you can now work with all the functions available in the package.

Computing Descriptive Statistics

One such function is the describe() function. To learn how to use a function, use the command below. Information about the function will show in a different panel in your R studio.

```
?describe
```

Note that it only make sense to compute descriptive variables for numeric variables.

Computing Descriptive Statistics

Suppose you want to compute the descriptive statistics for GPA variable.

```
describe(Add$GPA)
```

```
##    vars  n mean   sd median trimmed  mad  min max range
## X1    1 88 2.46 0.86   2.63    2.49 0.93 0.67   4  3.33
```

This will provide a number of descriptive statistics. You can see sample size(n), mean, standard deviation (sd), median, minimum value (min), maximum value (max), range, skewness (skew), and kurtosis.

Remember, either of the following lines of code will provide the same output.

```
describe(Add[,'GPA'])

describe(Add[,6])
```

Computing Descriptive Statistics

You can compute the descriptive statistics for many variables. Suppose you want to compute the descriptive statistics for ADDSC, GPA, and IQ variables in this dataset. The version below will provide the descriptive statistics only for the listed variables.

```
describe(Add[,c('GPA','IQ','ADDSC')])
```

```
##       vars  n   mean    sd median trimmed   mad   min ma
## GPA      1 88   2.46  0.86   2.63    2.49  0.93  0.67
## IQ       2 88 100.26 12.98 100.00   99.67 13.34 75.00 13
## ADDSC    3 88  52.60 12.42  50.00   52.18 10.38 26.00  8
##         se
## GPA   0.09
## IQ    1.38
## ADDSC 1.32
```

```
# You can also use the code below for the same output:
# describe(Add[,c(2,5,6)])
```

Computing Descriptive Statistics

Sometimes, you may want to compute conditional statistics. For instance, suppose you want to compute the descriptive statistics of GPA for students with and without Social Problems separately. You can use describeBy() function from the psych package. First, learn more about this function:

```
?describeBy
```

Computing Descriptive Statistics

Then, compute the descriptive statistics of GPA for students with and without social problems.

```
describeBy(x     = Add$GPA,
           group = Add$SocProb.f)
```

```
##
##  Descriptive statistics by group
## group: No
##    vars  n mean   sd median trimmed  mad  min max range
## X1    1 78 2.47 0.89   2.55     2.5 0.93 0.67   4  3.33
## ------------------------------------------------------
## group: Yes
##    vars  n mean   sd median trimmed mad min max range  s
## X1    1 10 2.38 0.61   2.67    2.48 0.3   1   3     2 -
```

Computing Descriptive Statistics

In the previous slide, I use two arguments for describeBy() function.

- x = Add$GPA, specifies the outcome variable you want to compute desc. stats
- group = Add$Dropout.f, specifies the grouping variable

Computing Descriptive Statistics

You can do the same thing for multiple variables.

```
describeBy(x     = Add[,c('GPA','IQ','ADDSC')],
           group = Add$SocProb.f)
```

```
##
##  Descriptive statistics by group
## group: No
##       vars  n   mean    sd median trimmed   mad   min ma
## GPA      1 78   2.47  0.89   2.55    2.50  0.93  0.67
## IQ       2 78 101.49 12.95 102.00  100.94 13.34 79.00 13
## ADDSC    3 78  50.96 11.49  50.00   50.62 10.38 26.00  7
##         se
## GPA   0.10
## IQ    1.47
## ADDSC 1.30
## ------------------------------------------------------------
## group: Yes
##       vars  n  mean    sd median trimmed   mad min max r
```

Computing Descriptive Statistics

If you want to compute quartiles (25th percentile, 50th percentile, and 75th percentile), you can use the quantile() function. Suppose you want to compute Q1 (25th), Q2 (50th), and Q3 (75th) for GPA:

```
quantile(x     = Add$GPA,
         probs = c(.25,.50,.75))
```

```
##   25%   50%   75%
## 1.750 2.635 3.000
```

Computing Descriptive Statistics

If you want to compute the frequencies for categorical variables, you can use the table() function.

```
table(Add$Sex.f)
```

```
##
##   Male Female
##     51     31
```

This indicates that there are 51 individuals in the datasets coded as Male, and 31 individuals coded as Female.

Computing Descriptive Statistics

```
table(Add$SocProb.f)
```

```
##
##  No Yes
##  78  10
```

This indicates that there are 78 individuals with no social problem, and 10 individuals with social problem.

You can use table() function to get frequencies for numeric variables, but we don't recommend this when you have many possible values since the table will be large and messy.

Computing Descriptive Statistics

If you want to create z-scores for numeric variables you can use scale() function. Below syntax will create and append new variables for standardized scores for numeric variables in the data.

```
Add$GPA.z   <- scale(Add$GPA)

Add$IQ.z    <- scale(Add$IQ)

Add$ADDSC.z <- scale(Add$ADDSC)
```

Check the descriptive statistics for the standardized variables.

```
describe(Add[,c('GPA.z','IQ.z','ADDSC.z')])
```

```
##         vars  n mean sd median trimmed  mad   min  max r
## GPA.z      1 88    0  1   0.21    0.04 1.08 -2.07 1.79
## IQ.z       2 88    0  1  -0.02   -0.05 1.03 -1.95 2.83
## ADDSC.z    3 88    0  1  -0.21   -0.03 0.84 -2.14 2.61
##           se
```

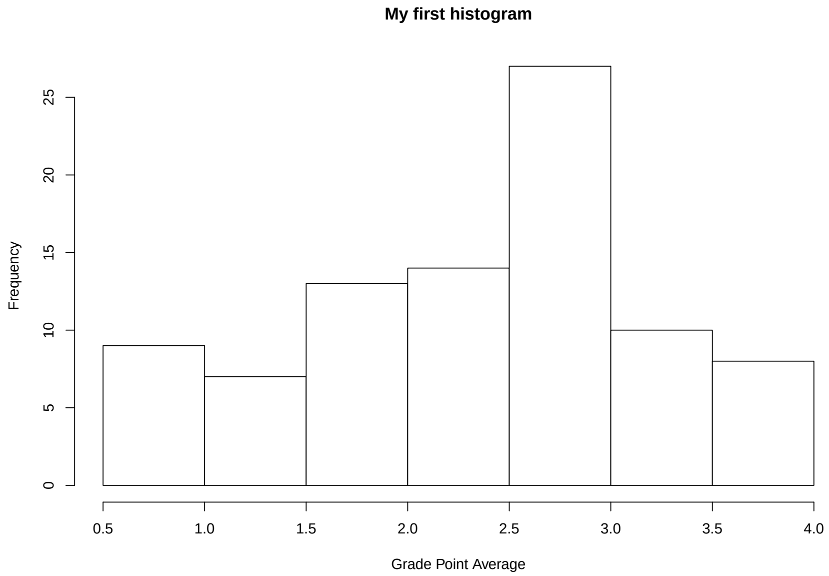
Histogram

Histogram

You can use hist() function to create a histogram and impose a normal density on it. For example, create a histogram of GPA.

```r
hist(x      = Add$GPA,
     freq   = TRUE,
     col    = 'white',
     border = 'black',
     xlab   = 'Grade Point Average',
     main   = 'My first histogram')
```

Histogram

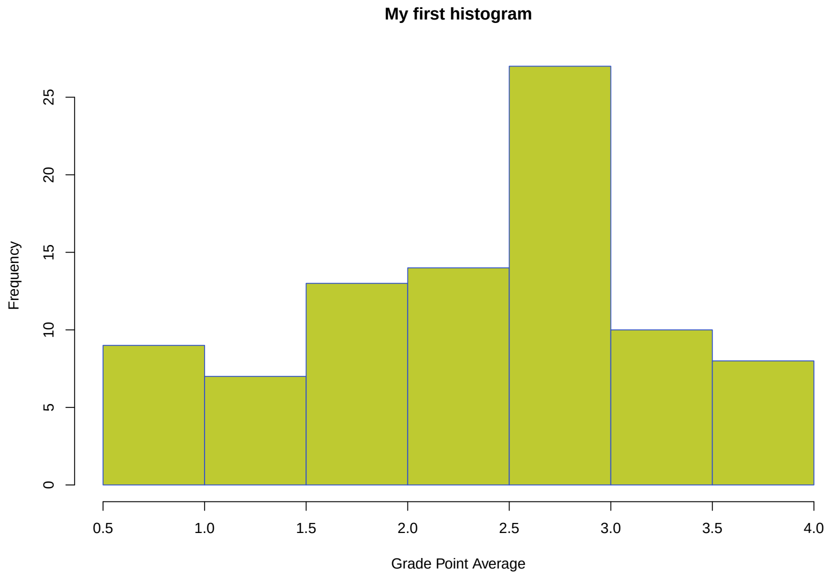

My first histogram
Frequency
0
5
10
15
20
25
0.5
1.0
1.5
2.0
2.5
3.0
3.5
4.0
Grade Point Average

Histogram

You can change the fill color and border color, you can use any HEX value: https://htmlcolorcodes.com/

```
hist(x      = Add$GPA,
     freq   = TRUE,
     col    = '#BECA31',
     border = '#3152CA',
     xlab   = 'Grade Point Average',
     main   = 'My first histogram')
```

Histogram



My first histogram
Frequency
0
5
10
15
20
25
0.5
1.0
1.5
2.0
2.5
3.0
3.5
4.0
Grade Point Average

Histogram

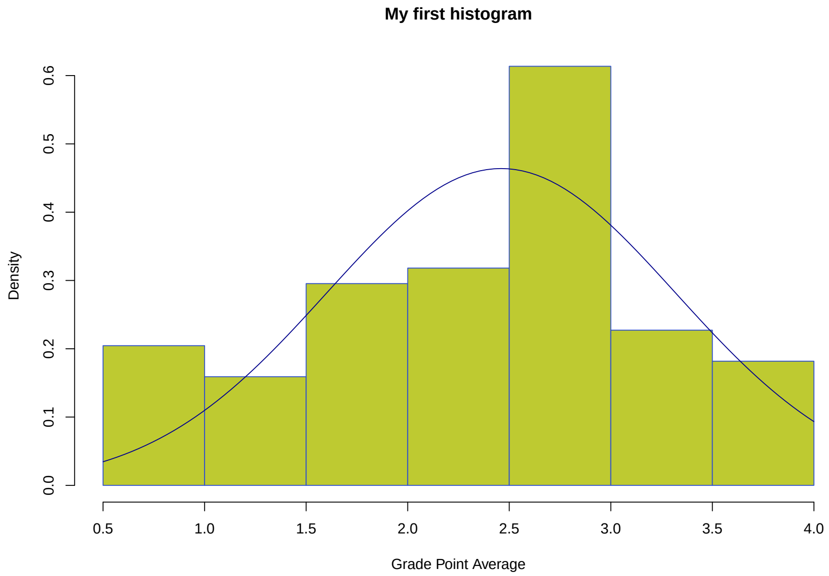
If you want to impose a normal curve on the histogram, first, you change the freq argument to FALSE. This modifies the Y-axis, so it doesn't represent frequency, and instead represent probability densities.

Then, add the normal density with the same mean and std obtained from GPA using the curve() function.

```r
hist(x      = Add$GPA,
     freq   = FALSE,
     col    = '#BECA31',
     border = '#3152CA',
     xlab   = 'Grade Point Average',
     main   = 'My first histogram')

curve(expr = dnorm(x, mean=2.46, sd=0.86),
             col  = "darkblue",
             add=TRUE)
```

Histogram


My first histogram
Density
0.0
0.1
0.2
0.3
0.4
0.5
0.6
0.5
1.0
1.5
2.0
2.5
3.0
3.5
4.0
Grade Point Average

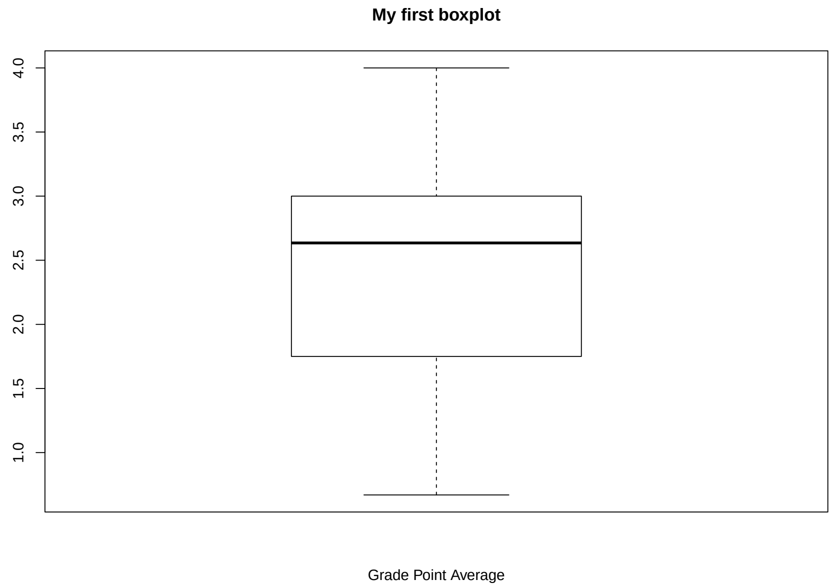
Boxplot

Boxplot

You can use boxplot() function to create a boxplot of, for example, GPA. See ?boxplot to see which values you can modify and add, like color, title, etc.

```
boxplot(x        = Add$GPA,
        main     = 'My first boxplot',
        xlab     = 'Grade Point Average',
        col      = 'white')
```

Boxplot


My first boxplot
Grade Point Average