Lab Tutorial 1: Basic R

Chris M. Fiacconi

2018-07-28

In this tutorial, you will learn some of the basics of the R programming language. R is an extremely useful tool for data manipulation, organization, analysis and reporting.

Get and Set Working Directory

When loading data into R, we need to know which directory (filepath) R thinks we're in. Otherwise, it won't be able to find the file we're looking for.

```
#Get working directory
currdir<-getwd()
print(currdir)
```

```
## [1] "/Users/chrisfiacconi/Dropbox/Teaching/PSYC 6940/Lab Tutorials/Lab Tutorial 1"
```

```
# We can also change the directory using the "Session" menu tab at the top of RStudio.
# By clicking this menu and then hovering over "Set Working Directory," we can select
# the folder containing the files of interest by clicking "Choose Directory."
```

Basic Arithmetic

We can do basic math within the R environment

```
5 + 10 # addition
```

```
## [1] 15
```

```
total<-5 + 10 # store the sum in a new variable called total (can be named whatever you like)
total + 10 # can add number to variable name (which we set to 15 above)
```

```
## [1] 25
```

```
32 - 11 # subtraction
```

```
## [1] 21
```

```
8*4 # multiplication
```

```
## [1] 32
```

```
144/12 #division
```

```
## [1] 12
```

```
sqrt(64) # square root of 64
```

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

```
numbers<-c(13,8,22,19,25) # join numbers together in a vector
mean(numbers) # calculate mean
```

```
## [1] 17.4
```

```
median(numbers) # calculate median
```

```
## [1] 19
```

```
sd(numbers) # calculate standard deviation
```

```
## [1] 6.8775
```

```
sum(numbers) # calculate sum
```

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

```
min(numbers) # get minimum value
```

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

```
max(numbers) # get maximum value
```

```
## [1] 25
```

```
length(numbers) # get number of elements in "numbers" variable
```

```
## [1] 5
```

Rounding In R

In this class, we'll adopt the practice of rounding non-integers to two decimal places. In your assignments, be sure to do this so you get the correct results!

```
example<-14.59284501
rnd_example<-round(example,digits=2)
print(rnd_example)
```

```
## [1] 14.59
```

```
# We could also combine the last two steps into one
print(round(example,digits=2))
```

```
## [1] 14.59
```

Loading Data Into R

Before we can organize, transform, or perform analyses on our data, we first need to know how to load a data file (in this course, the data files will be tab-delimited .txt files) into the R environment.

The **read.table** function can be used to load data files into R. The *header* argument indicates whether our data file has column names. In this case (and almost always in this course), the answer is yes, so we set it to "T" for TRUE. This data file lists the height of 40 students from UG and 20 students from Waterloo.

```
mydata<-read.table(file="heights.txt",header=T) # load in data file and give it a name
```

Often, we don't want to list the entire data file, as they can sometimes be quite large. If we just want to look at a portion of the data, we can used the **head** function, and specify the number of rows we want to see. Let's look at just the first 10 rows

```
head(mydata,n=10)
```

```
##    Subject School   Height
## 1        1 Guelph 171.1582
## 2        2 Guelph 143.4576
## 3        3 Guelph 170.3947
## 4        4 Guelph 187.2626
## 5        5 Guelph 174.8415
## 6        6 Guelph 173.7353
## 7        7 Guelph 192.0723
## 8        8 Guelph 184.4495
## 9        9 Guelph 213.3880
## 10      10 Guelph 178.7792
```

Let's take a closer look at the structure of our data using the **str** function (which stands for structure). This command will allow us to see how R sees our data.

```
str(mydata)
```

```
## 'data.frame':    80 obs. of  3 variables:
##  $ Subject: int  1 2 3 4 5 6 7 8 9 10 ...
##  $ School : Factor w/ 2 levels "Guelph","Waterloo": 1 1 1 1 1 1 1 1 1 1 ...
##  $ Height : num  171 143 170 187 175 ...
```

Our data variable is a *data.frame* which is a very common and useful type of data structure in R. It consists of multiple rows and columns, where each column can be a unique data type. For example, R identifies the School column as a *factor* with two levels, while the Subject and Height columns contain numeric data. Because the entries under the School column were text rather than numbers, R automatically thinks that this column identifies the different levels of an independent variable. The **str** function provides an excellent summary of how R structures our data.

If we want to access only one column within *mydata* (or any data frame), we can use the **$** symbol. Let's store the School column as its own variable, called schools.

```
schools<-mydata$School
```

We can also access particular rows and columns of a data frame by indexing with particular row and column numbers.

```
# General indexing method - mydata[row,column]
mydata[4,] # get 4th row, all columns
mydata[32,3] # get 32nd row, 3rd column
mydata[5:10,1:2] # get values from rows 5 to 10 in columns 1 and 2
```

Data Formatting with Tidyverse

Although the basic version of R contains many useful functions, there are external packages developed by users that are designed to make data wrangling and organization much simpler than the basic commands that are built-in to R. One such package is **tidyverse** which actually contains multiple sub-packages all designed to facilitate managing and summarizing your data.

Let's install and load **tidyverse**:

```
install.packages("tidyverse") # make sure you are connected to the internet
library(tidyverse) # load the package into the R environment
```

Now, lets separate the height values according to School, and put them into two separate variables using the **filter** function from the *dplyr* package that is part of the **tidyverse**.

```r
Guelph<-filter(mydata,School=="Guelph") # separate the values in
# the height column that come from Guelph students and store them in a new variable
# called Guelph

head(Guelph,n=5)
```

```
##   Subject School   Height
## 1       1 Guelph 171.1582
## 2       2 Guelph 143.4576
## 3       3 Guelph 170.3947
## 4       4 Guelph 187.2626
## 5       5 Guelph 174.8415
```

```r
Waterloo<-filter(mydata,School=="Waterloo") # separate the values in
#the height column that come from Waterloo students and store them in a new variable
#called Waterloo

head(Waterloo,n=5)
```

```
##   Subject   School   Height
## 1      41 Waterloo 189.7056
## 2      42 Waterloo 184.6231
## 3      43 Waterloo 186.8270
## 4      44 Waterloo 162.7278
## 5      45 Waterloo 175.6068
```

One of my favourite commands within the *dplyr* package is the **group_by** command. This function allows you to group your data by condition so that subsequent commands can perform functions on each group. This is extremely useful when you want to summarize your data according to different experimental conditions. Let's use this function to group the data within the *mydata* dataframe according to the School variable.

```r
by_school<-group_by(mydata,School) # the new variable "by_school" is now grouped according to school
```

Get Descriptive Statistics

Now that we grouped the data by School, we can summarize and get descriptive statistics for each condition using the **summarize** function, and specifying which summary statistics we want.

```r
school.descriptives<-summarize(by_school,Mean=mean(Height),S.Dev=sd(Height),n=length(Height))
print(school.descriptives)
```

```
## # A tibble: 2 x 4
##     School     Mean    S.Dev     n
##     <fctr>    <dbl>    <dbl> <int>
## 1   Guelph 182.3318 14.07352    40
## 2 Waterloo 185.3012 16.00613    40
```

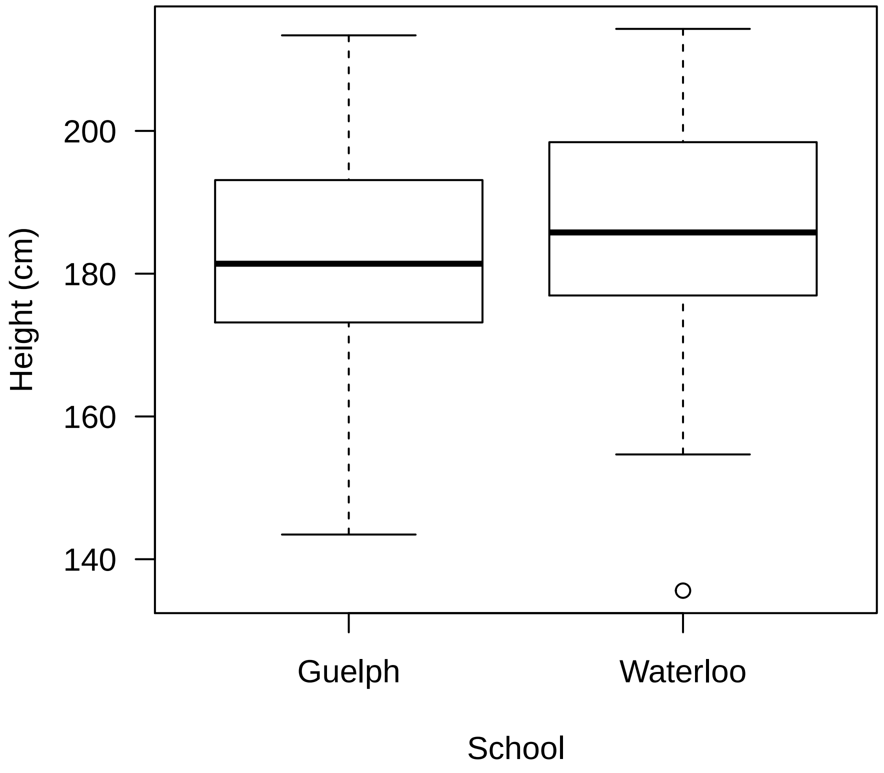
Plot/Visualize the Data

One of the most important skills in research is plotting or graphing your data. Plots are a convenient and simple way to communicate and share your results with others. R has many, many different plotting functions, and we'll learn a few in this class, but we'll start with a few simple ones right now.

Boxplot

Let's create a boxplot, which displays the inter-quartile (Q1-Q3) range of our data along with the median. Boxplots can provide us with a good idea of whether our data is normally distributed. We'll create two boxplots, one for each school.

```
boxplot(Height~School,data=mydata,las=1,ylab="Height (cm)",xlab="School")
```



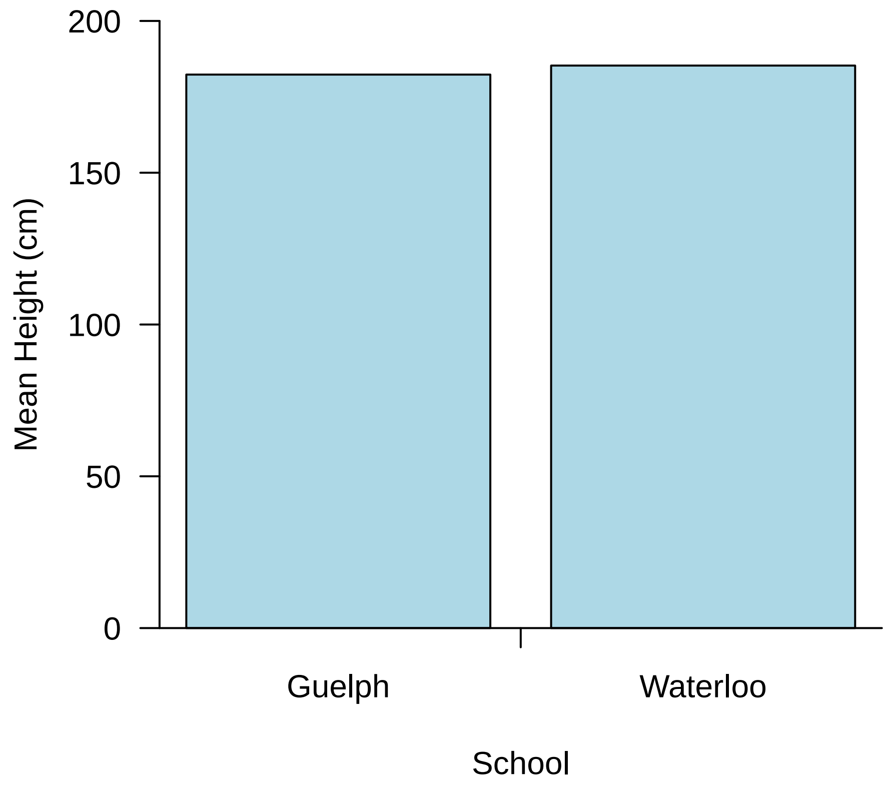
```
# You can save plots by clicking on the 'Export' button in the Plots tab in R Studio!
```

The *Height~School* term tells R that we want separate boxplots for our height data for each school. We also have to tell R that the column names Height and School can be found within the mydata variable. The other arguments are mostly concerned with aesthetics of the plot, including the orientation of the y-axis number labels, the colour of the boxplots, and the axis labels.

Bar Graph

We can also make a bar graph to plot the mean heights for each school:

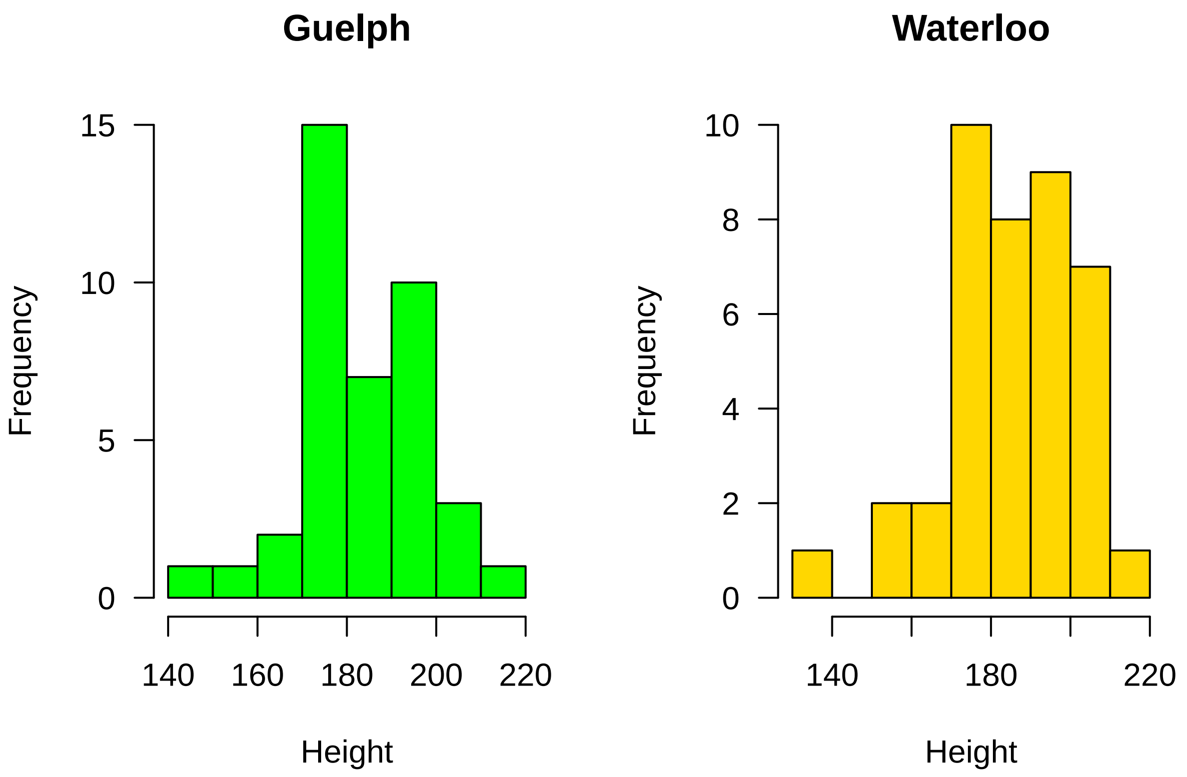
```
barplot(height=school.descriptives$Mean,names.arg=school.descriptives$School,beside=T,
        las=1,col="lightblue",ylab="Mean Height (cm)",xlab="School",ylim=c(0,200))

axis(side=1,at=c(.1,1.3,2.6),labels=c("","","")) # add in x-axis
```



Histogram

Finally, let's make a histogram of our height values according to school. A histogram displays the frequency with which particular data points occur in our sample, and they allow us to get a sense of how our data are distributed:

```r
par(mfrow=c(1,2)) # create two plot panels
hist(x=Guelph$Height,breaks=8,las=1,col="green",ylab="Frequency",xlab="Height",main="Guelph")
hist(x=Waterloo$Height,breaks=8,las=1,col="gold",ylab="Frequency",xlab="Height",main="Waterloo")
```



Going From the Raw Data to Descriptive Statistics

Most experiments in psychology require that you collect multiple observations from multiple different research participants. Therefore, your raw data will consist of values at the level of individual trials (each observation per participant). Before we can calculate the condition-level descriptive statistics (as we did above), we need to aggregate the data across trials for each participant. Fortunately, this task is made simple using the commands within the **tidyverse** packages. First, let's load in a representative data set.

```
raw.data<-read.table(file="EmoStroop.txt",header=T)
str(raw.data)
```

```
## 'data.frame':    5377 obs. of  4 variables:
##  $ Conditions : Factor w/ 2 levels "Congruent","Incongruent": 1 1 1 1 1 1 1 1 1 1 ...
##  $ TargValence: Factor w/ 2 levels "Negative","Positive": 2 2 2 2 2 2 2 2 2 2 ...
##  $ RT         : int  1445 1277 1098 1256 775 1326 940 1056 746 1212 ...
##  $ Subj       : int  1 1 1 1 1 1 1 1 1 1 ...
```

This dataset comes from an experiment that examined whether response conflict is perceived as emotionally 'negative'. On each trial, participants ($n = 30$) were primed with either a congruent or incongruent Stroop item (e.g., the word "blue" printed in green) followed by either a positive (e.g., "adore"), or negative (e.g., "murder") word. Thus, there were a total of four conditions in the experiment, and there were 48 trials in each condition. Reaction time (RT) classify the word as positive or negative was the dependent measure of interest.

Each row represents a trial, with the columns representing variables indicating the condition, the RT, and which participant was associated with a given trial. First, we need to aggregate across trials within each condition for each participant, deriving a summary statistic that captures the central tendency of the observations for that condition for that participant. Let's use the **group_by** function followed by the **summarize** function contained within the *dplyr* package to obtain these values:

```r
by_subj.cond<-group_by(raw.data,Subj,Conditions,TargValence) # Group the
# variables of interest into "chunks"
subj.medians<-summarize(by_subj.cond,medianRT=median(RT),nTrials=length(RT)) # Get median
# and no. trials for each condition for each participant

print(subj.medians)
```

```
## # A tibble: 120 x 5
## # Groups:   Subj, Conditions [?]
##     Subj  Conditions TargValence medianRT nTrials
##    <int>      <fctr>      <fctr>    <dbl>   <int>
##  1     1   Congruent    Negative    914.0      47
##  2     1   Congruent    Positive    801.0      46
##  3     1 Incongruent    Negative    899.0      44
##  4     1 Incongruent    Positive    822.5      44
##  5     2   Congruent    Negative    559.5      44
##  6     2   Congruent    Positive    551.0      43
##  7     2 Incongruent    Negative    548.0      45
##  8     2 Incongruent    Positive    528.0      41
##  9     3   Congruent    Negative    505.0      46
## 10     3   Congruent    Positive    528.0      45
## # ... with 110 more rows
```

The preceding code calculated the median RT for each participant for each of the four conditions, along with the number of trials within each condition for each participant (exclusion of outliers and incorrect trials resulted in <=48 trials per condition)

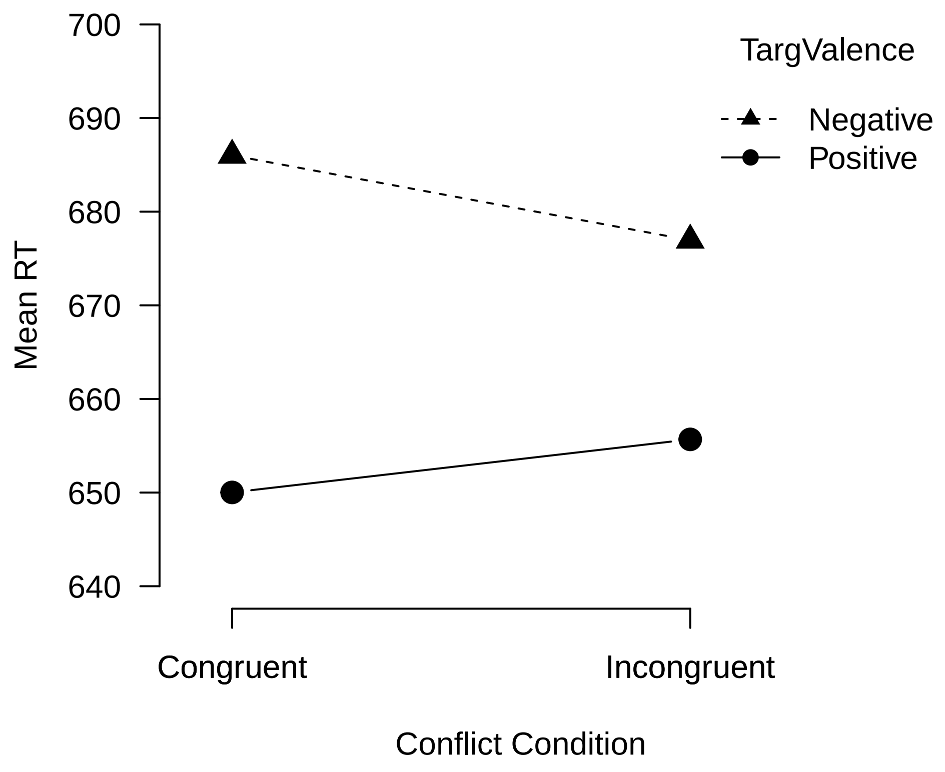
Now that we have the median RTs, let's aggregate across participants to derive the mean RT for each condition:

```r
by_conds<-group_by(subj.medians,Conditions,TargValence,add=FALSE) # make sure to set add=FALSE!!
overall.means<-summarize(by_conds,meanRT=mean(medianRT),nSubj=length(medianRT))
print(overall.means)
```

```
## # A tibble: 4 x 4
## # Groups:   Conditions [?]
##    Conditions TargValence   meanRT nSubj
##        <fctr>      <fctr>    <dbl> <int>
## 1   Congruent    Negative 686.0167    30
## 2   Congruent    Positive 650.0167    30
## 3 Incongruent    Negative 676.9500    30
## 4 Incongruent    Positive 655.6833    30
```

Now that we have the mean RTs for each condition, we can plot these means using the **interaction.plot** function to visualize the pattern of results:

```r
with(overall.means,interaction.plot(x.factor=Conditions,
                              trace.factor=TargValence,response=meanRT,las=1,
                              ylab="Mean RT",xlab="Conflict Condition",ylim=c(640,700),
                              bty="n",type="b",cex=1.5,pch=c(17,19))) # make plot, remove outer box

axis(side=1,at=c(1,2),labels=c("Congruent","Incongruent")) # add in x-axis with ticks and labels
```



Consistent with the idea that response conflict is inherently aversive, it appears as though negative words are responded to faster following an incongruent Stroop prime, with the opposite pattern holding for the positive words.

Long Vs. Wide Format

Another key aspect of data organization in R concerns the distinction between the *wide* and *long* data formats. To this point, the data have been organized in long format, which is the format that R prefers. In the long format, each row of the data frame is an observation, where the columns represent variables that take on different values across observations. However, depending on the design of the experiment (especially repeated-measures designs), the data may be initially organized in the wide format, where each row is a participant, and the columns represent scores among the different experimental conditions. Consider the following data set organized in wide format:

```
bpdrug.data<-read.table(file="bpdrug.txt",header=T)
head(bpdrug.data)
```

```
##   Subj    mg100    mg150    mg200
## 1    1 122.8915 130.2325 120.8630
## 2    2 129.1299 128.1279 116.6290
## 3    3 124.9678 124.0082 120.6927
## 4    4 129.3927 126.4388 118.4236
## 5    5 125.1873 123.4438 119.4742
## 6    6 133.3509 121.5304 111.1048
```

This data set contains the systolic blood pressure readings of 20 different participants who received each of three doses of a new blood pressure drug (100mg, 150mg, 200mg). The data are in *wide* format, with each dose level getting its own column and each row representing the observed BP reading for that participant. However, in order to utilize the majority of functions in R, the data need to be converted to long format. Fortunately, the *tidyr* package which comes with tidyverse contains the **gather** and **spread** functions which transform the data from wide to long, and from long to wide formats, respectively. Let's use the **gather** function to convert the data into long format:

```
bpdata.long<-gather(bpdrug.data,key="Dose",value="SysBP",mg100:mg200)
head(bpdata.long)
```

```
##   Subj  Dose    SysBP
## 1    1 mg100 122.8915
## 2    2 mg100 129.1299
## 3    3 mg100 124.9678
## 4    4 mg100 129.3927
## 5    5 mg100 125.1873
## 6    6 mg100 133.3509
```

Now that the data are in long format, we can use the built-in functions within R to calculate descriptive statistics using the **group_by** and **summarize** functions that we used above.

```
by_dose<-group_by(bpdata.long,Dose)
dose.means<-summarize(by_dose,meanBP=mean(SysBP),sdBP=sd(SysBP),nSubj=length(SysBP))
print(dose.means)
```

```
## # A tibble: 3 x 4
##    Dose   meanBP     sdBP nSubj
##   <chr>    <dbl>    <dbl> <int>
## 1 mg100 128.2321 3.889886    20
## 2 mg150 126.8867 3.322595    20
## 3 mg200 117.9121 5.355847    20
```