# data_orgmarkdown

R uses <- for variable assignment. The variable "subjects" is being assigned a list of numbers (vector) from 501 to 551. The variable "data_path" is being assigned the location of the data set. Then, setwd is used to set the working directory.

```
subjects <- c(501:551)
data_path <- "/Volumes/home/RAs/Azara/R_dataorg/data/"
setwd(data_path)
```

In R, functions come as part of a package, so you must make sure the necessary packages are installed. The # before the function, in this case, renders it a comment rather than a piece of code. This is because you don't want to install packages multiple times, but it is here for demonstrative purposes. If you don't know what packages you need, Google it! In the following example 'plyr' is the name of the package:

```
#install.packages('plyr')
#install.packages('plotrix')
```

Use "library" to specify which packages you are using. This will direct R to the functions you want to use.

```
library(plyr)
```

```
## Warning: package 'plyr' was built under R version 3.2.5
```

```
library(reshape)
```

```
## 
## Attaching package: 'reshape'
```

```
## The following objects are masked from 'package:plyr':
## 
##     rename, round_any
```

```
library(reshape2)
```

```
## 
## Attaching package: 'reshape2'
```

```
## The following objects are masked from 'package:reshape':
## 
##     colsplit, melt, recast
```

```
library(data.table)
```

```
## 
## Attaching package: 'data.table'
## The following objects are masked from 'package:reshape2':
##
##     dcast, melt
```

```
## The following object is masked from 'package:reshape':
##
##     melt
```

```
library(plotrix)
```

```
## Warning: package 'plotrix' was built under R version 3.2.5
```

We include this with a "/" because when we define the variable "path" later on, we need to have a "/" between subject number and recog.csv.

```
recog_file="/recog.csv"
```

This is a for statement that runs 51 times (once for each subject). The variable "sub_num" is the subject number. The data set we are considering has quite a few empty spaces that we do not want to include in our analysis. We use the function "paste0" to concatenate everything without any spaces in between, and we assign that to the variable "path". For example, for subject 501, "path" is defined as /Volumes/home/RAs/Azara/R_dataorg/data/501/recog.csv We then assign a variable "recog" to read the .csv file into R. "recog$X <- NULL" deletes the extra column. The if statement says that if the loop is on its first run, put the output from the for statement into the variable "recog_all". If the loop is on any other run, bind the subsequent data frames by rows. Note: if you wanted to bind the data frames by columns, you would use "cbind" instead of "rbind".

```
for (sub in 1:length(subjects)) {
  sub_num <- subjects[sub]
  path <- paste0(data_path,sub_num,recog_file)
  recog <- read.csv(path)
  recog$X <- NULL       # Remove mystery column "X"
  if (sub==1){
    recog_all <- recog
  } else {
    recog_all <- rbind(recog_all,recog)
  }
}
```

The "subset" function is used to select the data in the specified columns and place them in the recog_all variable. In the other option,"which" is used to select observations in recog_all.

```
# if you want a bunch of columns and you know the column headers, then you can do it as:
recog_all <- subset(recog_all,select=-c(preTrial_recog_acc,preTrial_source_acc,preStimTrial_recog_acc,preStimTrial_source_acc))
 # OR
```

```r
recog_all[,-which(names(recog_all) %in%
c("preTrial_recog_acc","preTrial_source_acc","preStimTrial_recog_acc","preStimTrial_source_acc"))]
```

```
## data frame with 0 columns and 19992 rows
```

Straightforward way to delete columns:

```r
# if you want to delete a bunch of columns, and you know the column numbers,
then you can do it as:
recog_all <- recog_all[-c(19:22)]  # if you want to delete columns 19 through
22.
```

Creating a new column called "recog_all$delay" for the inter-stimulus interval; the 1* turns false to zero and true to one.

```r
# Add a new column for ISI
recog_all$delay <- 1*(recog_all$iti==4)     # short ISI = 0; long ISI = 1
```

Another way to add a new column where responses are either zero or one (new or old).

```r
# Add a new column for old/new judgments (whether subjects judged the stim as
old or new)
recog_all$OldResp[recog_all$recog_resp_bin==1] <- 0
recog_all$OldResp[recog_all$recog_resp_bin>1] <- 1
# ^^ can also be written as:  recog_all$OldResp <-
1*(recog_all$recog_resp_bin>1)
```

Straightforwardly converting the values in column old_num into either ones or zeros using 1* (if the extant value is equal to 1, it remains the same, if not it becomes 0).

```r
# convert old_num values into 1's and 0's (currently, old=1 & new=2)
recog_all$old_num <- 1*(recog_all$old_num==1)    # now, old=1 & new=0
```

In data set, we had "nan" when a participant missed a response; we do not want these to be counted in our analysis, so we have to make them "NA" instead.

```r
# correcting psychopy errors* (actually, my errors in psychopy!)
recog_all$recog_resp[recog_all$recog_resp=='nan'] <- NA   # psychopy was
treating 'nan' as a text!
recog_all$recog_resp_bin[is.na(recog_all$recog_resp)] <- NA   # psychopy had
coded the response bins for non-responses as 0 instead of NA
```

The variable "names" now contains the columns specified in the brackets. The function lapply goes over a set of data and returns a list. Within the subset of data labelled names that we have assigned to the variable recog_all, we will receive a list of our columns as factors or categories.

```r
# changing a bunch of columns into factors
names <-
c('sub_num','old_num','trial_num','block_num','stim_num','word_num','enc1_trial','preWord','iti','next_iti','delay')
```

```
recog_all[,names] <- lapply(recog_all[,names],factor)
# recog_all$sub_num <- as.factor(recog_all$sub_num)
```

Straightforward ways to rename a column:

```
# if you want to rename a column name
names(recog_all)[names(recog_all)=="enc1_trial"] <- 'enc_trial'
# if you know the column number, ^ can be easily done as:
names(recog_all)[9] <- 'enc_trial'
```

Here, we are renaming iti and next_iti with isi and next_isi respectively. Two ways of renaming functions:

```
# rename function
setnames(recog_all,old=c("iti","next_iti"),new=c("isi","next_isi"))  #
data.table library
# OR
stats <- rename(recog_all,replace=c("iti"="isi","next_iti"="next_isi"))   #
plyr library
```

The function ddply takes the data you have, splits it up, performs some action on it, and then returns it to your data frame. In this case, we use ddply to centralize reaction times for our subjects, and assign these values into our variable "recog_all".

```
# if you want to centralize reaction time
recog_all <- ddply(recog_all, c("sub_num"), transform, RT_cent =
scale(recog_RT))
```

The cast function organizes our data in rows, with each row having a unique identifier.

```
#### OVERALL ####
rt <- cast(recog_all,value="recog_RT", sub_num~old_num, mean,na.rm=T)
rt_stim <- cast(recog_all,value="recog_RT", sub_num~stim_num, mean,na.rm=T,
margins=T)
```

Here, we are using cast to organize the data in the columns specified in the brackets, then using the name function on stats to set the names of this object.

```
# dprime, criteria, hits, FA
stats <- cast(recog_all,value="OldResp", sub_num~old_num, mean,na.rm=T)
names(stats) <- c('subject','FA','hits')
```

Here, in rows with no false alarms, we are setting it so that 1 out of 197 trials is expected to be a false alarm.

```
# Changing FA for rows which contain 0 FA to 1/197 (expecting the next
(197th) trial to be a false alarm)
no_FA <- which(stats$FA==0)
for (i in no_FA) stats[i,2] <- 1/197
```

Here, in rows with all hits, we are setting it so that one trial is expected to be a miss.

```
# Changing hits for rows which contain all hits to 196/197 (expecting the
next (197th) trial to be a miss)
perfect_hits <- which(stats$hits==1)
for (i in perfect_hits) stats[i,3] <- 196/197
```

Here we calculate the sensitivity index (d'prime) and assigning it to stat$dprime. In normally distributed data, the qnorm function finds the boundary of the area under the curve; by subtracting the boundary between hits and false alarms, we find d'prime.

```
# Calculating d-prime and criteria
stats$dprime <- qnorm(stats$hits)-qnorm(stats$FA)
stats$criteria <- -0.5*(qnorm(stats$hits)+qnorm(stats$FA))
```

Once we find d'prime and criteria, we use the following functions to compute basic statistics on our results.

```
mean(stats$dprime)

## [1] 3.046511

std.error(stats$dprime)

## [1] 0.09667004

mean(stats$criteria)

## [1] 0.1146218

std.error(stats$criteria)

## [1] 0.04516179

mean(stats$hits)

## [1] 0.905558

std.error(stats$hits)

## [1] 0.01039161

mean(stats$FA)

## [1] 0.07508273

std.error(stats$FA)

## [1] 0.01740739
```

We use the t.test function to perform a t-test on our data.

```
t.test(stats$dprime,mu=0)

##
##  One Sample t-test
##
## data:  stats$dprime
## t = 31.515, df = 50, p-value < 2.2e-16
## alternative hypothesis: true mean is not equal to 0
## 95 percent confidence interval:
##  2.852343 3.240678
## sample estimates:
## mean of x
##  3.046511
```