PSYC 6780: Lab Tutorial 2

Chris M. Fiacconi

Indexing with Logical Statements

In the previous tutorial, we selected elements of a vector based on their index or position within the vector. It is also helpful to be able to select out certain elements of a vector that meet a given criterion. We can do this by using logical statements within the [] bracket notation.

```r
my_numbers<-c(3,7,1,2,9,3,4)
my_numbers2<-c(1,4,5,2,4,1,5)

# Get numbers > 3
greater_3<-my_numbers[my_numbers > 3]
print(greater_3)
```

```
## [1] 7 9 4
```

```r
# Get numbers != 3
not_equal_3<-my_numbers[my_numbers != 3]
print(not_equal_3)
```

```
## [1] 7 1 2 9 4
```

```r
# Combine logical statement with function
mean(my_numbers2)
```

```
## [1] 3.142857
```

```r
mean_subset<-my_numbers[my_numbers > mean(my_numbers2)]
print(mean_subset)
```

```
## [1] 7 9 4
```

```r
# Create data.frame and index rows with logical statement
my_numbers_data<-data.frame(set_1=my_numbers,set_2=my_numbers2)
rows_mean_subset<-my_numbers_data[my_numbers_data[,1] > mean(my_numbers_data[,2]),]
print(rows_mean_subset)
```

```
##   set_1 set_2
## 2     7     4
## 5     9     4
## 7     4     5
```

Data Wrangling and Summarizing 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**:

```r
install.packages("tidyverse") # make sure you are connected to the internet - only do this once!
library(tidyverse) # load the package into the R environment - do this each time you open R!
```

```
## Warning: package 'ggplot2' was built under R version 3.5.2
```

We'll learn to use the following functions from *tidyverse* to wrangle and summarize our data:

```r
filter() # select particular rows
group_by() # chunk data together for easy summarizing
summarize() # summarize grouped data
arrange() # sort data
select() # choose a specific variable (column) by name
mutate() # add a column to existing data frame
```

Let's now load the .txt file from Week 1 into R. Remember, 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 40 students from Waterloo.

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

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.

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

# Same thing using the 'pipe' (%>%)
by_school<-mydata %>% group_by(School)

# Notice that the data frame always goes first here - the %>% symbol is a way to chain together differe
# functions on the same data set. You can think of it as meaning, 'and then.'
```

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.

```
options(pillar.sigfig = 6) # Control the number of significant figures output - need this only once!

school.descriptives<-mydata %>% group_by(School) %>%
  summarize(Mean=mean(Height),S.Dev=sd(Height),n=n())

print(school.descriptives)
```

```
## # A tibble: 2 x 4
##   School      Mean   S.Dev     n
##   <fct>      <dbl>   <dbl> <int>
## 1 Guelph   182.332 14.0735    40
## 2 Waterloo 185.301 16.0061    40
```

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:

```
subj.medians<-raw_data %>% group_by(Subj,Conditions,TargValence) %>%
  summarize(medianRT=median(RT),nTrials=n()) # Get median and no. trials for each condition
# for each participant.
# the n() function will tell us how many elements are in each chunk

print(subj.medians)
```

```
## # A tibble: 120 x 5
## # Groups:   Subj, Conditions [?]
##     Subj Conditions  TargValence medianRT nTrials
##    <int> <fct>       <fct>          <dbl>   <int>
##  1     1 Congruent   Negative       914        47
##  2     1 Congruent   Positive       801        46
##  3     1 Incongruent Negative       899        44
##  4     1 Incongruent Positive       822.5      44
##  5     2 Congruent   Negative       559.5      44
##  6     2 Congruent   Positive       551        43
##  7     2 Incongruent Negative       548        45
##  8     2 Incongruent Positive       528        41
##  9     3 Congruent   Negative       505        46
## 10     3 Congruent   Positive       528        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:

```
by_cond_means<-subj.medians %>% group_by(Conditions,TargValence,add=FALSE) %>%
  summarize(meanRT=mean(medianRT),nSubj=n()) # make sure to set add=FALSE!!

print(by_cond_means)
```

```
## # A tibble: 4 x 4
## # Groups:   Conditions [?]
##   Conditions  TargValence  meanRT nSubj
##   <fct>       <fct>         <dbl> <int>
## 1 Congruent   Negative    686.017    30
## 2 Congruent   Positive    650.017    30
## 3 Incongruent Negative    676.95     30
## 4 Incongruent Positive    655.683    30
```

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 - Reformatting Data

One-Way Repeated-Measures Design

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)
str(bpdata_long) # See structure of data frame
```

```
## 'data.frame':    60 obs. of  3 variables:
##  $ Subj : int  1 2 3 4 5 6 7 8 9 10 ...
##  $ Dose : chr  "mg100" "mg100" "mg100" "mg100" ...
##  $ SysBP: num  123 129 125 129 125 ...
```

```r
head(bpdata_long) # Look at first 6 rows of new data frame
```

```
##   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.

```r
by_dose_means<-bpdata_long %>% group_by(Dose) %>%
  summarize(meanBP=mean(SysBP),sdBP=sd(SysBP),nSubj=n())

print(by_dose_means)
```

```
## # A tibble: 3 x 4
##   Dose   meanBP    sdBP nSubj
##   <chr>   <dbl>   <dbl> <int>
## 1 mg100 128.232 3.88989    20
## 2 mg150 126.887 3.32259    20
## 3 mg200 117.912 5.35585    20
```

Two-Way Repeated-Measures Design

The previous example was fairly simple, as the experiment consisted of a one-way design. Let's look at how to transform data from wide to long with a more complex 2 × 2 design.

```r
angle_data<-read.table(file="Ch12T1.txt",header=T) # Load data file
head(angle_data)
```

```
##   Absent0 Absent4 Absent8 Present0 Present4 Present8
## 1     420     420     480      480      600      780
## 2     420     480     480      360      480      600
## 3     480     480     540      660      780      780
## 4     420     540     540      480      780      900
## 5     540     660     540      480      660      720
## 6     360     420     360      360      480      540
```

Now we will convert the data from wide format to long format using the **gather** function together with the **separate** function.

```r
angle_data$Subject<-as.factor(1:nrow(angle_data)) # add Subject ID column to angle.data
angle_long<-gather(data=angle_data,key=Noise.Angle,value=RT,-Subject) # Gather columns except Subject
head(angle_long)
```

```
##   Subject Noise.Angle  RT
## 1       1     Absent0 420
## 2        2     Absent0 420
## 3        3     Absent0 480
## 4        4     Absent0 420
## 5        5     Absent0 540
## 6        6     Absent0 360
```

```
# Break up Noise.Angle column into 2 separate columns
angle_long<-separate(angle_long,col=Noise.Angle,into=c("Noise","Angle"),sep=-1)

angle_long$Noise<-as.factor(angle_long$Noise) # Ensure Noise is treated as a factor!!
angle_long$Angle<-as.factor(angle_long$Angle) # Ensure Angle is treated as a factor!!

head(angle_long,n=12)
```

```
##    Subject  Noise Angle  RT
## 1        1 Absent     0 420
## 2        2 Absent     0 420
## 3        3 Absent     0 480
## 4        4 Absent     0 420
## 5        5 Absent     0 540
## 6        6 Absent     0 360
## 7        7 Absent     0 480
## 8        8 Absent     0 480
## 9        9 Absent     0 540
## 10      10 Absent     0 480
## 11       1 Absent     4 420
## 12       2 Absent     4 480
```

```
# Convert back from long to wide format with unite and spread functions
angle_long_unite<-unite(angle_long,col="Noise_Angle",Noise,Angle)
angle_data_wide<-spread(angle_long_unite,key=Noise_Angle,value=RT)

print(angle_data_wide)
```

```
##    Subject Absent_0 Absent_4 Absent_8 Present_0 Present_4 Present_8
## 1        1      420      420      480       480       600       780
## 2        2      420      480      480       360       480       600
## 3        3      480      480      540       660       780       780
## 4        4      420      540      540       480       780       900
## 5        5      540      660      540       480       660       720
## 6        6      360      420      360       360       480       540
## 7        7      480      480      600       540       720       840
## 8        8      480      600      660       540       720       900
## 9        9      540      600      540       480       720       780
## 10      10      480      420      540       540       660       780
```

More Tools For Structuring and Re-organizing your data

Now, let's see how we can use the **arrange**, **select**, and **mutate** commands with the *bpdata* data set to help organize and add additional structure to the data set.

```r
by_subj_num<-bpdata_long %>% arrange(Subj) # Order dataframe by Subj (increasing)
by_subj_num_desc<-bpdata_long %>% arrange(desc(Subj)) # Order dataframe by Subj (decreasing)
head(by_subj_num)
```

```
##   Subj  Dose    SysBP
## 1    1 mg100 122.8915
## 2    1 mg150 130.2325
## 3    1 mg200 120.8630
## 4    2 mg100 129.1299
## 5    2 mg150 128.1279
## 6    2 mg200 116.6290
```

```r
# We can also sort by more than one variable
by_subj_sys<-bpdata_long %>% arrange(Subj,SysBP)
head(by_subj_sys)
```

```
##   Subj  Dose    SysBP
## 1    1 mg200 120.8630
## 2    1 mg100 122.8915
## 3    1 mg150 130.2325
## 4    2 mg200 116.6290
## 5    2 mg150 128.1279
## 6    2 mg100 129.1299
```

```r
# To isolate one column, we can use the select() function
dose<-bpdata_long %>% select(Dose)
head(dose)
```

```
##    Dose
## 1 mg100
## 2 mg100
## 3 mg100
## 4 mg100
## 5 mg100
## 6 mg100
```

```r
# If you want the values/labels only without the title use pull()
dose_values<-bpdata_long %>% pull(Dose)
head(dose_values)
```

```
## [1] "mg100" "mg100" "mg100" "mg100" "mg100" "mg100"
```

```r
# We can add a new column to our dataframe using the mutate() function
gender<-rep(c("Male","Female"),times=10,each=3) # Create gender variable

bpdata_long_gender<-by_subj_num %>% mutate(Gender=gender) # Append gender variable
head(bpdata_long_gender)
```

```
##   Subj  Dose    SysBP Gender
## 1    1 mg100 122.8915   Male
## 2    1 mg150 130.2325   Male
## 3    1 mg200 120.8630   Male
## 4    2 mg100 129.1299 Female
## 5    2 mg150 128.1279 Female
## 6    2 mg200 116.6290 Female
```

Loading Multiple Data Files and Combining Them

So far, the raw data files we've loaded into *R* (e.g., EmoStroop.txt) have contained the data from all subjects. However, you will often collect one data file at a time, and it is very helpful to load all of these data files at the same time and combine them into a single data file for easy analysis in *R*. Let's learn how to do this with a series of .txt files.

```
# Set wd to folder where data files are located
setwd("/Users/chrisfiacconi/Dropbox/Teaching/PSYC 6780/Lab Tutorial 2/Subject Data Files")

file_names<-dir(pattern = ".txt") # Get names of all files that match the .txt extension

# apply the read.table function to each file
files<-lapply(file_names,read.table,sep = "\t", fill = TRUE,header = TRUE)

files_combined<-do.call(rbind,files) # Combine all files into a single data frame

# Now let's analyze the data
str(files_combined)
```

```
## 'data.frame':    22 obs. of  3 variables:
##  $ Subject  : int  1 1 1 1 1 1 1 1 1 1 ...
##  $ Condition: Factor w/ 3 levels "c1","c2","Practice": 3 3 1 2 1 2 1 2 1 2 ...
##  $ RT       : int  500 550 450 560 480 620 550 430 600 590 ...
```

```
files_combined$Subject<-as.factor(files_combined$Subject) # Convert Subject column to factor

exp_trials<-files_combined %>% filter(Condition != "Practice") # Get rid of practice trials

by_subj_descriptives<-exp_trials %>% group_by(Subject,Condition) %>%
  summarize(MeanRT=mean(RT),S.Dev=sd(RT),n=n()) # Get means/SD/num_trials for each subject

print(by_subj_descriptives)
```

```
## # A tibble: 4 x 5
## # Groups:   Subject [?]
##   Subject Condition MeanRT    S.Dev     n
##   <fct>   <fct>      <dbl>    <dbl> <int>
## 1 1       c1         540    73.8241     5
## 2 1       c2         550    83.6660     4
## 3 2       c1         590    73.4847     5
## 4 2       c2         547.5 107.819      4
```

```
overall_means<-by_subj_descriptives %>% group_by(Condition, add=FALSE) %>%
  summarize(meanRT=mean(MeanRT)) # Get overall means for each condition

print(overall_means)
```

```
## # A tibble: 2 x 2
##   Condition meanRT
##   <fct>      <dbl>
## 1 c1         565
## 2 c2         548.75
```

Dealing with Missing Data

Unfortunately, data does not always come in a neat package. For various reasons, it is not uncommon to have missing values in your data due to factors such as attrition, errors, etc. Fortunately, *R* has a number of tools that make sorting and re-organizing data with missing values fairly straightforward. Typically (although not always), these missing values are represented by an NA value.

```r
# Create data with missing values

stocks<-tibble(
  year = c(2015, 2015, 2015, 2015, 2016, 2016, 2016),
  qtr = c(1,2,3,4,2,3,4),
  return = c(1.88,0.59,0.35,NA,0.92,0.17,2.66)
)

head(stocks)
```

```
## # A tibble: 6 x 3
##    year   qtr return
##   <dbl> <dbl>  <dbl>
## 1  2015     1   1.88
## 2  2015     2   0.59
## 3  2015     3   0.35
## 4  2015     4  NA
## 5  2016     2   0.92
## 6  2016     3   0.17
```

```r
# Identify NAs
is.na(stocks$return)
```

```
## [1] FALSE FALSE FALSE  TRUE FALSE FALSE FALSE
```

```r
which(is.na(stocks$return) == TRUE)
```

```
## [1] 4
```

Notice that there are actually two missing values here - one explicit (NA) and one implicit (2016, qtr #1). A very useful function that allows you to identify all missing values (implicit and explicit) is the **complete** function. This function will identify all unique combinations of row values within the columns you specify so that you can see any missing values that may be present.

```r
stocks_complete<-stocks %>%
  complete(year, qtr)

print(stocks_complete)
```

```
## # A tibble: 8 x 3
##    year   qtr return
##   <dbl> <dbl>  <dbl>
## 1  2015     1   1.88
## 2  2015     2   0.59
## 3  2015     3   0.35
## 4  2015     4  NA
## 5  2016     1  NA
## 6  2016     2   0.92
## 7  2016     3   0.17
## 8  2016     4   2.66
```

```r
# If you try to use a function when NAs are present, you'll get NA as output
mean(stocks$return)
```

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

```r
# You can avoid this by telling mean to ignore NAs
mean(stocks$return,na.rm=TRUE)
```

```
## [1] 1.095
```

```r
# Get rid of rows with NA values
stocks_clean<-stocks_complete %>% filter(!is.na(return))

print(stocks_clean)
```

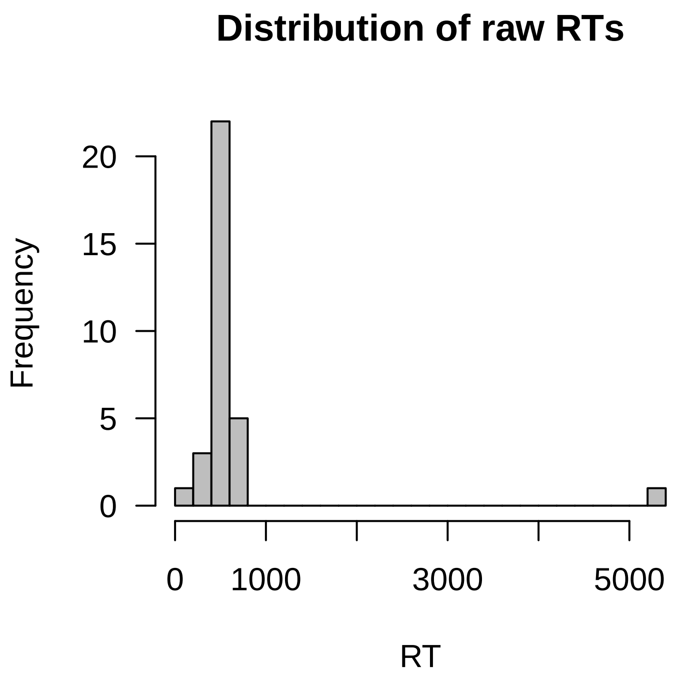
```
## # A tibble: 6 x 3
##    year   qtr return
##   <dbl> <dbl>  <dbl>
## 1  2015     1   1.88
## 2  2015     2   0.59
## 3  2015     3   0.35
## 4  2016     2   0.92
## 5  2016     3   0.17
## 6  2016     4   2.66
```

Dealing with Outliers

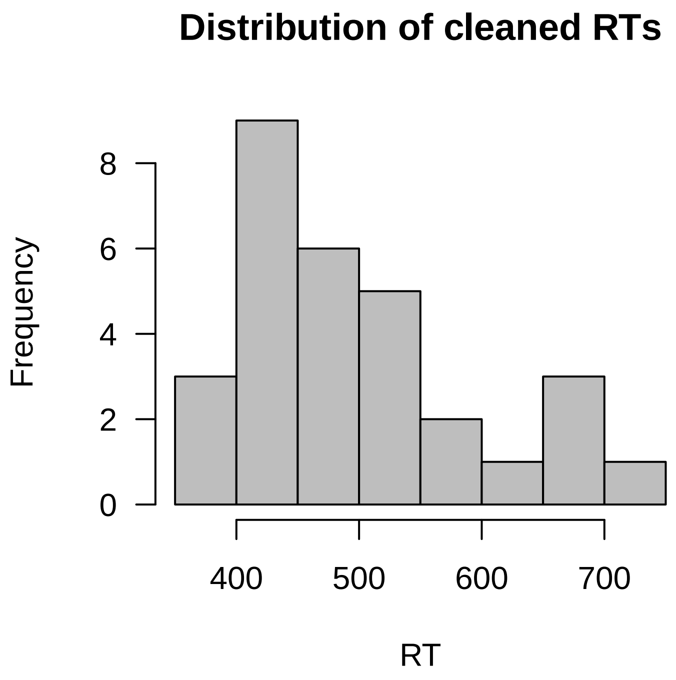
Outliers can have a dramatic influence on your data, especially if you are primarily interested in mean levels of performance. There are many different approaches to dealing with outliers, but here we will focus on how to identify and remove outliers from a dataset containing reaction times (RTs).

Absolute Cut-offs

```r
# Basic approach - remove RTs < 200 ms and > 5000 ms
rts<-as.integer(rnorm(n=30,mean=500,sd=90)) # Create data without outliers
rts_with_noise<-c(rts,150,5390) # add in outliers
rts_with_noise<-data.frame(Subject=seq(1,32,1),RT=rts_with_noise)
```



```r
# Two equivalent methods
clean_rts<-filter(rts_with_noise,RT > 200 & RT < 5000)
clean_rts<-filter(rts_with_noise,RT > 200,RT < 5000) # Same as above
```



Relative Cut-offs

Although this is a reasonable strategy, it is often not desirable to use absolute RT cutoffs only. Often, we may want to exclude RTs that are some number of standard deviations above or below the mean. Let's learn how to do this.

```r
sd_cutoff<-2.5 # Set SD criterion for cutoff

sd_clean_rts<-rts_with_noise %>%
  filter(RT > (mean(RT)-sd(RT)*sd_cutoff), RT < (mean(RT)+sd(RT)*sd_cutoff))
```

This method is easy to implement and works well. However, often we want to exclude outliers in a given condition based on the mean and SD of RTs within *each cell* of our experimental design. To accomplish this, we need to chunk the data and group it by condition before applying the filtering procedure. Let's use the data from the *EmoStroop* data set.

```r
str(raw_data) # Look at structure of EmoStroop 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 ...
```

Recall, these data come from a 2 × 2 factorial design with Condition (Congruent, Incongruent) and TargValence (Negative, Positive) as independent variables.

```r
sd_cutoff<-2 # Use 2 SD as criterion

raw_ntrials<-raw_data %>%
  group_by(Subj,Conditions,TargValence) %>%
  summarize(nTrials=n()) # Get n trials per cell

print(raw_ntrials)
```

```
## # A tibble: 120 x 4
## # Groups:   Subj, Conditions [?]
##     Subj Conditions  TargValence nTrials
##    <int> <fct>       <fct>         <int>
##  1     1 Congruent   Negative         47
##  2     1 Congruent   Positive         46
##  3     1 Incongruent Negative         44
##  4     1 Incongruent Positive         44
##  5     2 Congruent   Negative         44
##  6     2 Congruent   Positive         43
##  7     2 Incongruent Negative         45
##  8     2 Incongruent Positive         41
##  9     3 Congruent   Negative         46
## 10     3 Congruent   Positive         45
## # ... with 110 more rows
```

```r
emo_clean_rts<-raw_data %>%
  group_by(Subj,Conditions, TargValence) %>%
  filter(RT > (mean(RT)-sd(RT)*sd_cutoff), RT < (mean(RT)+sd(RT)*sd_cutoff)) # Remove outliers

clean_ntrials<-emo_clean_rts %>%
  group_by(Subj,Conditions,TargValence) %>%
  summarize(nTrials=n()) # Look at n trials per cell after removing outliers

print(clean_ntrials)
```

```
## # A tibble: 120 x 4
## # Groups:   Subj, Conditions [?]
##     Subj Conditions  TargValence nTrials
##    <int> <fct>       <fct>         <int>
##  1     1 Congruent   Negative         44
##  2     1 Congruent   Positive         45
##  3     1 Incongruent Negative         41
##  4     1 Incongruent Positive         43
##  5     2 Congruent   Negative         42
##  6     2 Congruent   Positive         41
##  7     2 Incongruent Negative         44
##  8     2 Incongruent Positive         39
##  9     3 Congruent   Negative         45
## 10     3 Congruent   Positive         41
## # ... with 110 more rows
```

```
num_rawtrials<-sum(raw_ntrials$nTrials) # Get total number of raw trials
num_cleantrials<-sum(clean_ntrials$nTrials)# Get total number of remaining clean trials

prop_removed<-(num_rawtrials-num_cleantrials)/num_rawtrials # Calculate proportion of trials removed
print(prop_removed)
```

```
## [1] 0.0524456
```

Non-Recursive Moving Criteria

Although choosing a fixed SD criterion is a common approach, an even better approach is to choose a variable SD criterion that is based on the number of trials in each cell of the design. Although many experiments have equal numbers of trials in each condition, the number can often be unequal due to exclusion of error trials and for experiments in which there is a manipulation of the proportion of trials across conditions.

This approach was introduced by VanSelst & Joliceour (1994), who came up with a series of SD criteria that vary based on cell size. Essentially, they created a look-up table that indicates which SD criteria to use given the cell size in a given condition. Let's first load in the lookup table:

```
SD_criteria<-read.table("joliceour_outlier.txt",header=TRUE)
head(SD_criteria,n=10)
```

```
##    celln criterion
## 1      1     1.458
## 2      2     1.458
## 3      3     1.458
## 4      4     1.458
## 5      5     1.680
## 6      6     1.841
## 7      7     1.961
## 8      8     2.050
## 9      9     2.120
## 10    10     2.173
```

Now let's try and implement this procedure using the *tidyverse* tools we've covered to date. We'll work with the *PrimingRTs.txt* data file.

```
priming_data<-read.table(file="PrimingRTs.txt",header=TRUE)
str(priming_data)
```

```
## 'data.frame':    1837 obs. of  4 variables:
##  $ Subject   : int  73 73 73 73 73 73 73 73 73 73 ...
##  $ Word      : Factor w/ 180 levels "adopts","ammo",..: 31 73 36 104 157 20 140 74 76 30 ...
##  $ WordType  : Factor w/ 2 levels "Primed","Unprimed": 2 1 2 1 1 2 2 1 1 2 ...
##  $ JOLWord.RT: int  2082 2875 3308 3361 1569 12117 793 2672 2232 3032 ...
```

```
# How many subjects?
n_sub<-length(unique(priming_data$Subject))
print(n_sub)
```

```
## [1] 36
```

```
# Get n trials for each condition for each subject and raw means per conditions
priming_data %>%
  group_by(Subject,WordType) %>%
  summarize(nTrials=n(),meanRT=mean(JOLWord.RT))
```

```
## # A tibble: 72 x 4
## # Groups:   Subject [?]
##    Subject WordType nTrials  meanRT
##      <int> <fct>      <int>   <dbl>
##  1      73 Primed        25 2415.72
##  2      73 Unprimed      26 3279.54
##  3      74 Primed        27 2217.78
##  4      74 Unprimed      28 1556.93
##  5      75 Primed        28 2134.93
##  6      75 Unprimed      27 2704.07
##  7      76 Primed        30 1817.2
##  8      76 Unprimed      28 2091.29
##  9      77 Primed        29 1711.79
## 10      77 Unprimed      26 1292.08
## # ... with 62 more rows
```

```
# Get rid of outliers using VanSelst & Joliceour (1994) procedure
clean_priming_rts<-priming_data %>%
  group_by(Subject,WordType) %>%
  filter(JOLWord.RT > (mean(JOLWord.RT)-sd(JOLWord.RT)*SD_criteria$criterion[n()])) %>%
  filter(JOLWord.RT < (mean(JOLWord.RT)+sd(JOLWord.RT)*SD_criteria$criterion[n()]))

num_raw_priming_trials<-dim(priming_data)[1]
num_clean_priming_trials<-dim(clean_priming_rts)[1]

prop_removed<-(num_raw_priming_trials-num_clean_priming_trials)/(num_raw_priming_trials)
print(prop_removed)
```

```
## [1] 0.03538378
```

```r
# Get n trials for each condition for each subject and cleaned means per conditions
clean_priming_rts %>%
  group_by(Subject,WordType) %>%
  summarize(nTrials=n(),meanRT=mean(JOLWord.RT))
```

```
## # A tibble: 72 x 4
## # Groups:   Subject [?]
##    Subject WordType nTrials  meanRT
##      <int> <fct>      <int>   <dbl>
##  1      73 Primed        24 2124.5
##  2      73 Unprimed      25 2926.04
##  3      74 Primed        26 2002.19
##  4      74 Unprimed      27 1466.11
##  5      75 Primed        27 2044.22
##  6      75 Unprimed      26 2437.38
##  7      76 Primed        29 1715
##  8      76 Unprimed      28 2091.29
##  9      77 Primed        28 1627.93
## 10      77 Unprimed      25 1190.44
## # ... with 62 more rows
```

Binning Data

Another useful technique is to analyze the data in bins - that is, broken down into pieces (usually quartiles). When examining RTs for two conditions, it can be informative to see whether the difference between conditions is consistent across the entire distribution of RTs (i.e., the distribution is shifted), or whether any differences are confined to the tails of the distribution.

Let's divide the RT data into quartiles and plot the mean RT for each quartile. This can easily be achieved using the **cut_number** function which divides the data into n bins of equal size.

```r
nBins<-4

# Create and add column for quartile bins
RT_data_quart<-clean_priming_rts %>%
  group_by(Subject,WordType) %>%
  mutate(bin=cut_number(JOLWord.RT,n=nBins))

head(RT_data_quart)
```

```
## # A tibble: 6 x 5
## # Groups:   Subject, WordType [2]
##   Subject Word   WordType JOLWord.RT bin
##     <int> <fct>  <fct>         <int> <chr>
## 1      73 clever Unprimed       2082 (1.13e+03,2.23e+03]
## 2      73 hand   Primed         2875 (2.7e+03,5.4e+03]
## 3      73 cruel  Unprimed       3308 (2.23e+03,4.39e+03]
## 4      73 nymphs Primed         3361 (2.7e+03,5.4e+03]
## 5      73 tasty  Primed         1569 (1.33e+03,1.67e+03]
## 6      73 river  Unprimed        793 [708,1.13e+03]
```

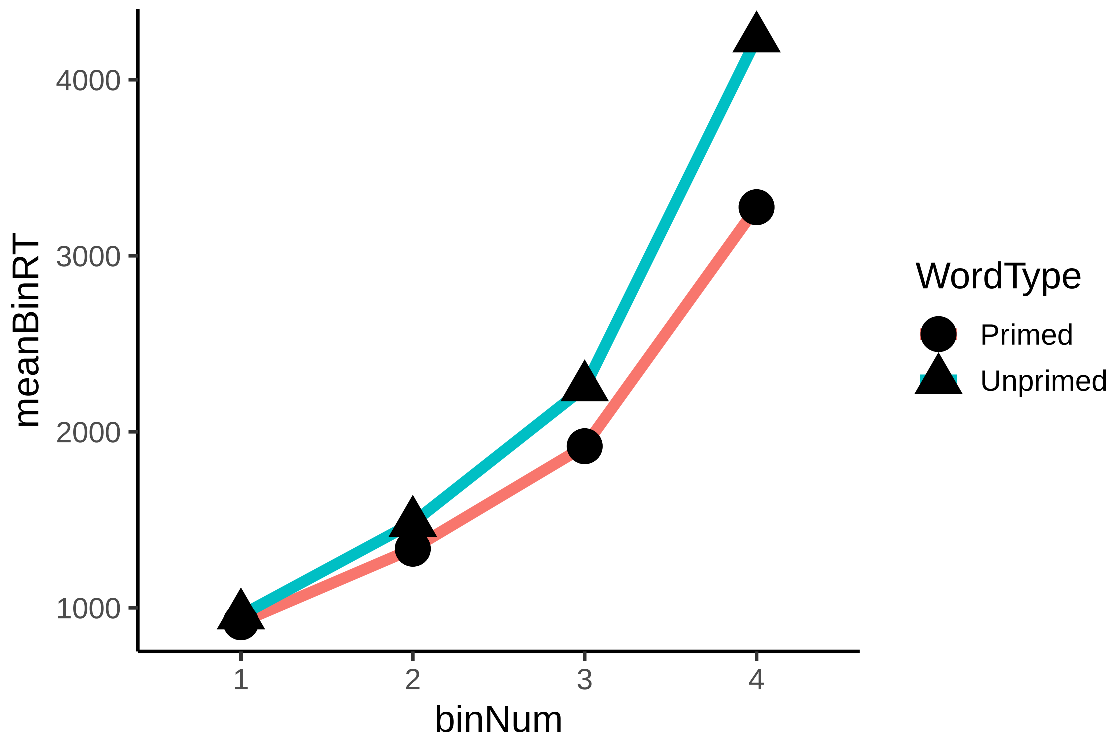
```r
# Get subject-level means for each bin and create binNum column
meanRT_quartiles<-RT_data_quart %>%
  group_by(Subject,WordType,bin,add=FALSE) %>%
  summarize(meanRT=mean(JOLWord.RT),nObs=n()) %>%
  arrange(Subject,WordType,meanRT) %>%
  mutate(binNum=rep(seq(1,4,1)))

meanRT_quartiles$binNum<-as.factor(meanRT_quartiles$binNum) # Convert binNum to factor for plotting

# Get overall mean RT for each bin
overall_meanRT_byquart<-meanRT_quartiles %>%
  group_by(WordType,binNum,add=FALSE) %>%
  summarize(meanBinRT=mean(meanRT))

print(overall_meanRT_byquart)
```

```
## # A tibble: 8 x 3
## # Groups:   WordType [?]
##   WordType binNum meanBinRT
##   <fct>    <fct>      <dbl>
## 1 Primed   1          917.774
## 2 Primed   2         1335.56
## 3 Primed   3         1918.06
## 4 Primed   4         3275.86
## 5 Unprimed 1          956.843
## 6 Unprimed 2         1483.00
## 7 Unprimed 3         2252.69
## 8 Unprimed 4         4234.87
```


meanBinRT
binNum
WordType
Primed
Unprimed