Lab 1: R Basics

Wednesday Bushong

Adapted from Anne Pier Salverda's materials for CSP 519 Spring 2016

1 Preliminaries

In this lab, we will go over some basic functions that can be used to inspect and handle data in R. For more information about any of the functions used, type "?function", for instance, "?mean".

2 Basics

Like most interpreted languages, you can type expressions directly for evaluation like so:

```
1 + 1
```

```
## [1] 2
```

However, you'll probably want to assign things to variables!

```
x <- 1 + 1
```

The assignment operator in R is <- instead of =. You can use the equals sign sometimes and R will accept it, but *do not fall into the habit of doing that!!!!*

Note that the output of a variable assignment is supressed by default (unlike in Matlab, which some of you may know). If you want to see output of variable you can just print it. Alternatively, when you're making the variable assignment you can wrap the entire expression in (parentheses).

```
x
```

```
## [1] 2
```

```
print(x)
```

```
## [1] 2
```

```
(y <- 3 + 4)
```

```
## [1] 7
```

Here's a quick rundown of the logical operators which will become very useful in the future:

```
1 == 1
```

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

```r
1 != 2
```

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

```r
1 == 1 & 1 == 2
```

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

```r
1 == 1 | 1 == 2
```

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

```r
xor(1 == 1, 1 != 2)
```

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

Quick overview of vectors and matrices:

```r
## Creating and indexing vectors
myvec <- c(1, 2, 3, 4)
myvec
```

```
## [1] 1 2 3 4
```

```r
myvec[1]
```

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

```r
myvec[1:3]
```

```
## [1] 1 2 3
```

```r
myvec[c(1, 3)] # return first and third element of vector
```

```
## [1] 1 3
```

```r
myvec[myvec == 1] # return all of the elements in the vector that have a value of "1"
```

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

```r
## Standard functions for summarizing data
mean(myvec)
```

```
## [1] 2.5
```

```r
sd(myvec)
```

```
## [1] 1.290994
```

```r
range(myvec)
```

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

```r
min(myvec)
```

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

```r
max(myvec)
```

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

```r
## Creating and indexing in matrices
mymat <- matrix(c(1:10, 21:30), ncol = 2)
mymat
```

```
##       [,1] [,2]
##  [1,]    1   21
##  [2,]    2   22
##  [3,]    3   23
##  [4,]    4   24
##  [5,]    5   25
##  [6,]    6   26
##  [7,]    7   27
##  [8,]    8   28
##  [9,]    9   29
## [10,]   10   30
```

```r
mymat[3, 2]
```

```
## [1] 23
```

```r
mymat[3, ] # leaving an index blank means "all"
```

```
## [1]  3 23
```

```r
mean(mymat[, 2]) # get mean of the second column
```

```
## [1] 25.5
```

```r
mymat[mymat[, 1] == 5, ] # all the rows which have a "5" in the first column
```

```
## [1]  5 25
```

These are all operations that will come in handy. Now let's get to the data type that will be the meat of this class: data frames! These are essentially like matrices that have named columns

```
fake.data <- data.frame(subject = c(rep(1, 3), rep(2, 3), rep(3, 3)),
                        RT = c(300, 400, 500, 320, 440, 450, 250, 300, 400),
                        Condition = rep(c("Condition 1", "Condition 2", "Condition 3"), 3))
```

Now, you can still use the same matrix indexing that we saw before:

```
fake.data[, 3]
```

```
## [1] Condition 1 Condition 2 Condition 3 Condition 1 Condition 2 Condition 3
## [7] Condition 1 Condition 2 Condition 3
## Levels: Condition 1 Condition 2 Condition 3
```

However, the data.frame data structure is awesome because you can view columns using the names of columns instead!

```
fake.data$Condition # does same thing as last line
```

```
## [1] Condition 1 Condition 2 Condition 3 Condition 1 Condition 2 Condition 3
## [7] Condition 1 Condition 2 Condition 3
## Levels: Condition 1 Condition 2 Condition 3
```

Data.frames also have lots of handy built-in functions that make subsetting much easier. For example, the subset() function allows you to easily access rows of a data frame that have desired values in particular columns:

```
fake.data[fake.data$Condition == "Condition 1", ] # eww, ugly!
```

```
##   subject  RT   Condition
## 1       1 300 Condition 1
## 4       2 320 Condition 1
## 7       3 250 Condition 1
```

```
subset(fake.data, Condition == "Condition 1") # much easier!
```

```
##   subject  RT   Condition
## 1       1 300 Condition 1
## 4       2 320 Condition 1
## 7       3 250 Condition 1
```

2 Reading data

Getting to your file's location

Before trying to read in data, make sure that your data files are in your working directory, and if not, to point them to the correct one. You can do this either by specifying an entire filepath, or by the command *setwd()* (you can check current working directory with *getwd()*).

```
getwd()
```

```
## [1] "/Users/wbushong/Dropbox/Courses/CSP519_SP17/wbushong_labs/Lab1"
```

```
setwd("~/Dropbox/Courses/CSP519_SP17/wbushong_labs/Lab1")
```

If you're using RStudio, you can also set the working directory to the source file location in the GUI:

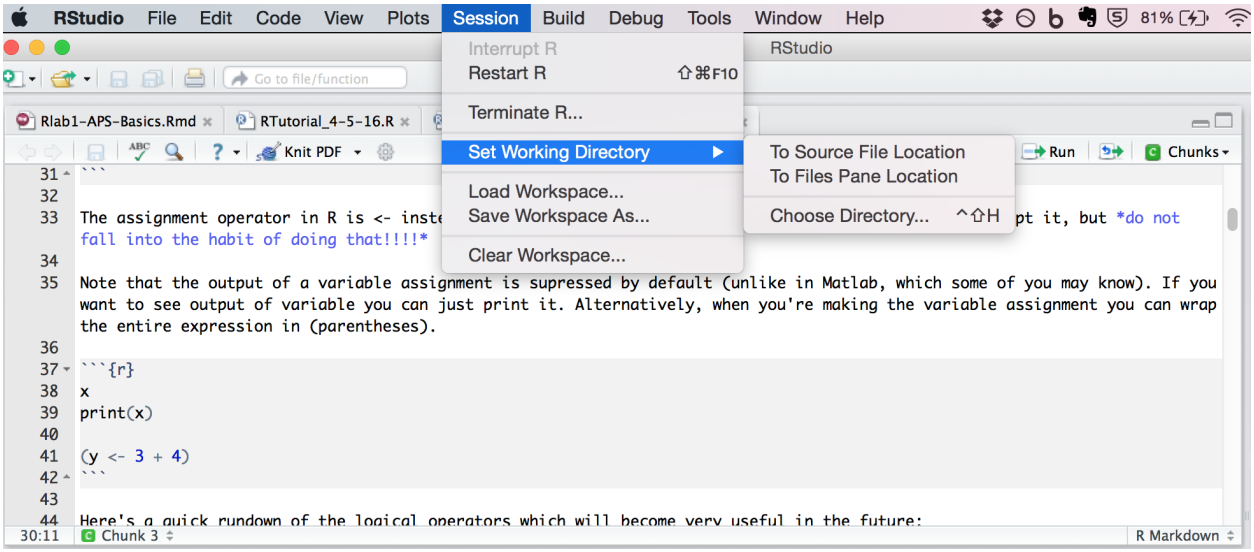

Figure 1: Setting working directory through the RStudio GUI.

Reading Data

If your data is in plain text format with comma-separated values, you can read it in using the *read.csv* function (see also the related functions *read.table* and *read.delim*).

```
boys <- read.csv("theboys.csv")
boys
```

```
##     name     instrument year_of_birth
## 1 George    lead guitar          1943
## 2   John  rhythm guitar          1940
## 3   Paul           bass          1942
## 4  Ringo         drums          1940
```

We will often use SPSS datafiles, which can be read with the *read.spss* function from the *foreign* package. Make sure to use the argument "to.data.frame = TRUE"; otherwise, the data will not be stored in a useable format. (The function often returns a warning, which you can ignore for the datasets used in this course.)

```
library(foreign) # library that allows you to read in .sav files
example <- read.spss("ExampleData1.sav", to.data.frame = TRUE)
```

3 Inspecting data in a dataframe

The *dim* function can be used to obtain the dimensions of the dataframe (number of rows; number of columns). See also the related functions *nrow* and *ncol*.

```
dim(example)
```

```
## [1]  7 17
```

The functions *head* and *tail* can be used to inspect just the first or last couple of rows in a dataframe.

```
head(example, 3)
```

```
##   ID UCS1 UCS2 UCS3 UCS4 UCS5 UCS6 UCS7 UCS8 UCS9 SOS1 SOS2 SOS3 SOS4 Age
## 1  1    4    1    5    4    3    3    5    2    5    4    7    2    3  18
## 2  2    1   NA    1    2    3    2    1    4   NA    6    4    3    2  18
## 3  3    4    1    5    4    3    3    5    2    5    6    4    3    3  23
##   Gender RelStatus
## 1      F         1
## 2      F         1
## 3      M         1
```

More generally, we can obtain partial information from a dataframe by specifying the relevant row and column indexes. This is just like the indexing we saw before for matrices!

```
example[1, ]                    # first row (all columns)
example[2:3, ]                  # second through third row
example[, 1]                    # first column (all rows)
example[, -(2:10)]              # exclude columns two through ten
example[1, c("Gender", "Age")] # returns values for gender and age for first case
example[1, c(16, 15)]           # yields identical result
```

The *summary* function provides basic descriptive statistics. For numerical values, the mean, minimum, maximum, and quartiles (including the median); for categorical values, a frequency count.

```
summary(example)
```

```
##        ID            UCS1            UCS2         UCS3         UCS4
##  Min.   :1.0   Min.   :1.000   Min.   :1   Min.   :1   Min.   :2.000
##  1st Qu.:2.5   1st Qu.:1.000   1st Qu.:1   1st Qu.:1   1st Qu.:2.000
##  Median :4.0   Median :1.000   Median :3   Median :3   Median :2.000
##  Mean   :4.0   Mean   :2.286   Mean   :3   Mean   :3   Mean   :2.857
##  3rd Qu.:5.5   3rd Qu.:4.000   3rd Qu.:5   3rd Qu.:5   3rd Qu.:4.000
##  Max.   :7.0   Max.   :4.000   Max.   :5   Max.   :5   Max.   :4.000
##                                NA's   :1   NA's   :1
##       UCS5         UCS6            UCS7         UCS8            UCS9
##  Min.   :3   Min.   :2.000   Min.   :1   Min.   :2.000   Min.   :2.0
##  1st Qu.:3   1st Qu.:2.000   1st Qu.:1   1st Qu.:2.000   1st Qu.:2.0
##  Median :3   Median :2.000   Median :3   Median :4.000   Median :3.5
##  Mean   :3   Mean   :2.429   Mean   :3   Mean   :3.143   Mean   :3.5
##  3rd Qu.:3   3rd Qu.:3.000   3rd Qu.:5   3rd Qu.:4.000   3rd Qu.:5.0
```

```
##  Max.   :3   Max.   :3.000   Max.   :5   Max.   :4.000   Max.   :5.0
##                              NA's   :1                   NA's   :1
##       SOS1            SOS2        SOS3          SOS4           Age
##  Min.   :2.000   Min.   :1   Min.   :2.0   Min.   :1.000   Min.   :18.00
##  1st Qu.:3.000   1st Qu.:3   1st Qu.:3.0   1st Qu.:2.250   1st Qu.:18.00
##  Median :6.000   Median :4   Median :3.0   Median :3.000   Median :19.00
##  Mean   :4.857   Mean   :4   Mean   :3.8   Mean   :3.167   Mean   :20.43
##  3rd Qu.:6.500   3rd Qu.:5   3rd Qu.:5.0   3rd Qu.:4.500   3rd Qu.:22.00
##  Max.   :7.000   Max.   :7   Max.   :6.0   Max.   :5.000   Max.   :26.00
##                              NA's   :2     NA's   :1
##  Gender RelStatus
##  F:3    0:3
##  M:4    1:4
##
##
##
##
##
```

Note that ID is interpreted as a numerical variable. This doesn't make sense (participant 2 is not twice participant 1). Therefore, we convert ID to a categorical variable using the function *as.factor*. The function *str* can be used to check the structure of the dataframe and verify that ID is now stored as a factor.

```
example$ID <- as.factor(example$ID)
str(example)
```

```
## 'data.frame':    7 obs. of  17 variables:
##  $ ID       : Factor w/ 7 levels "1","2","3","4",..: 1 2 3 4 5 6 7
##  $ UCS1     : num  4 1 4 1 4 1 1
##  $ UCS2     : num  1 NA 1 5 1 5 5
##  $ UCS3     : num  5 1 5 NA 5 1 1
##  $ UCS4     : num  4 2 4 2 4 2 2
##  $ UCS5     : num  3 3 3 3 3 3 3
##  $ UCS6     : num  3 2 3 2 3 2 2
##  $ UCS7     : num  5 1 5 NA 5 1 1
##  $ UCS8     : num  2 4 2 4 2 4 4
##  $ UCS9     : num  5 NA 5 2 5 2 2
##  $ SOS1     : num  4 6 6 7 7 2 2
##  $ SOS2     : num  7 4 4 3 6 3 1
##  $ SOS3     : num  2 3 3 NA NA 6 5
##  $ SOS4     : num  3 2 3 NA 1 5 5
##  $ Age      : num  18 18 23 19 21 26 18
##  $ Gender   : Factor w/ 2 levels "F","M": 1 1 2 1 2 2 2
##  $ RelStatus: Factor w/ 2 levels "0","1": 2 2 2 1 1 2 1
##  - attr(*, "variable.labels")= Named chr
##   ..- attr(*, "names")= chr
##  - attr(*, "codepage")= int 1252
```

Basic statistics like the mean, median, variance, and standard deviation can be obtained by using the functions *mean*, *median*, *var*, and *sd* on a variable.

```r
mean(example$Age)
```

```
## [1] 20.42857
```

```r
median(example$Age)
```

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

```r
var(example$Age)
```

```
## [1] 9.619048
```

```r
sd(example$Age)
```

```
## [1] 3.101459
```

The *subset* function can be used to select or analyze a subset of the data.

```r
subset(example, Age > 18)
```

```
##   ID UCS1 UCS2 UCS3 UCS4 UCS5 UCS6 UCS7 UCS8 UCS9 SOS1 SOS2 SOS3 SOS4 Age
## 3  3    4    1    5    4    3    3    5    2    5    6    4    3    3  23
## 4  4    1    5   NA    2    3    2   NA    4    2    7    3   NA   NA  19
## 5  5    4    1    5    4    3    3    5    2    5    7    6   NA    1  21
## 6  6    1    5    1    2    3    2    1    4    2    2    3    6    5  26
##   Gender RelStatus
## 3      M         1
## 4      F         0
## 5      M         0
## 6      M         1
```

```r
mean(subset(example, Gender == "M")$Age)
```

```
## [1] 22
```

The *rowMeans* function is useful for calculating the mean across several variables.

```r
rowMeans(example[, c("SOS1", "SOS2", "SOS3", "SOS4")])
```

```
## [1] 4.00 3.75 4.00   NA   NA 4.00 3.25
```

4 Missing values (NA's)

The *summary* and *str* functions revealed that there are missing values. For some statistical analyses, you can not use cases with missing values. The function *complete.cases* can be used to examine cases with NA's, and to create a dataframe with incomplete cases removed.

```
example[!complete.cases(example), ] # "!" means "not"
```

```
##   ID UCS1 UCS2 UCS3 UCS4 UCS5 UCS6 UCS7 UCS8 UCS9 SOS1 SOS2 SOS3 SOS4 Age
## 2  2    1   NA    1    2    3    2    1    4   NA    6    4    3    2  18
## 4  4    1    5   NA    2    3    2   NA    4    2    7    3   NA   NA  19
## 5  5    4    1    5    4    3    3    5    2    5    7    6   NA    1  21
##   Gender RelStatus
## 2      F         1
## 4      F         0
## 5      M         0
```

```
example_complete <- example[complete.cases(example), ]
```

Let's verify that the new dataframe has the right (expected) number of cases.

```
dim(example_complete)
```

```
## [1]  4 17
```

```
dim(example)
```

```
## [1]  7 17
```

5 Adding cases or variables

You can add new cases to a dataframe using the *rbind* function. Note that the two dataframes need to have the same (types of) variables.

```
psych_majors <- read.spss("PsychMajors.sav", to.data.frame = TRUE)
dim(psych_majors)
```

```
## [1] 10  6
```

```
head(psych_majors)
```

```
##   ID        SR1        SR2        SR3 Major GPA
## 1  1 -1.6436536 -1.8767581 -1.9269593 Psych 2.1
## 2  2 -1.0149369 -1.4984423 -0.7994487 Psych 3.0
## 3  3 -0.9919321 -1.2907494 -0.6596552 Psych 3.9
## 4  4 -0.9654679 -1.2758175  1.8162398 Psych 2.5
## 5  5 -0.9494163 -1.1589751 -2.8198061 Psych 2.0
## 6  6 -0.9387778 -0.8208735 -0.2328901 Psych 2.6
```

```
other_majors <- read.spss("OtherMajors.sav", to.data.frame = TRUE)
dim(other_majors)
```

```
## [1] 10  6
```

```
all_majors <- rbind(psych_majors, other_majors)
dim(all_majors)
```

```
## [1] 20  6
```

The *merge* function can be used to add new variables.

```
gender <- read.spss("Gender.sav", to.data.frame = TRUE)
head(gender, 3)
```

```
##   ID Gender
## 1  1   male
## 2  2   male
## 3  3 female
```

```
all_majors <- merge(all_majors, gender)
head(all_majors)
```

```
##   ID        SR1        SR2        SR3 Major GPA Gender
## 1  1 -1.6436536 -1.8767581 -1.9269593 Psych 2.1   male
## 2  2 -1.0149369 -1.4984423 -0.7994487 Psych 3.0   male
## 3  3 -0.9919321 -1.2907494 -0.6596552 Psych 3.9 female
## 4  4 -0.9654679 -1.2758175  1.8162398 Psych 2.5   male
## 5  5 -0.9494163 -1.1589751 -2.8198061 Psych 2.0 female
## 6  6 -0.9387778 -0.8208735 -0.2328901 Psych 2.6 female
```

By default, the function merges the dataframes on the variables that they have in common (here, "ID"). If the relevant variables have different names in the two dataframes, you can specify which variables to match on using the arguments "by.x" and "by.y".

Adding New Columns from Computations

Sometimes, you want computations you do on a data frame to be stored in that data frame! For example, maybe instead of using the SR1 and SR2 variables separately, you want to use their mean instead. You can do that by getting the mean for each individual and adding it on as a separate column:

```
all_majors$Mean_SR1_SR2 <- rowMeans(all_majors[, c("SR1", "SR2")])
head(all_majors)
```

```
##   ID        SR1        SR2        SR3 Major GPA Gender Mean_SR1_SR2
## 1  1 -1.6436536 -1.8767581 -1.9269593 Psych 2.1   male   -1.7602059
## 2  2 -1.0149369 -1.4984423 -0.7994487 Psych 3.0   male   -1.2566896
## 3  3 -0.9919321 -1.2907494 -0.6596552 Psych 3.9 female   -1.1413407
## 4  4 -0.9654679 -1.2758175  1.8162398 Psych 2.5   male   -1.1206427
## 5  5 -0.9494163 -1.1589751 -2.8198061 Psych 2.0 female   -1.0541957
## 6  6 -0.9387778 -0.8208735 -0.2328901 Psych 2.6 female   -0.8798257
```

Or, you might want to split GPA categorically into above 3.0 and below 3.0:

```r
all_majors$GPA_Category <- ifelse(all_majors$GPA > 3, "High GPA", "Low GPA")
```

6 Writing data

Saving data in SPSS format is a sin[1]. We save the data in plain text format, with comma-separated values.

```r
write.csv(all_majors, "AllMajors.csv", row.names = FALSE)
```

We did not save the row names, since those are redundant with the variable "ID". If you open AllMajors.csv in a text editor, you can see that the variables "Major" and "Gender" were saved as string values, with quotation marks. This ensures that the variable will be stored as a factor when it is imported using *read.csv*.

7 Practice Problems

1. Compute a new variable named mean.sr equal to the average of variables SR1, SR2, and SR3

2. Calculate the mean and standard deviation of mean.sr

3. Calculate the mean of mean.sr for Psychology majors only

4. Calculate the mean of mean.sr for people with a GPA over of 3.0 or higher

[1] Anne Pier's words, not mine ;)