# Week 1 - R and Stats recap

Marwin Carmo

PSC 103B - Statistical Analysis of Psychological Data

UC DAVIS

# Who am I

- 1st Year Graduate Student in Quantitative Psychology
- MSc in Psychiatry at University of Sao Paulo, Brazil
- Advised by Dr. Philippe Rast
- Studying intraindividual variability

# Lab and homework dynamics

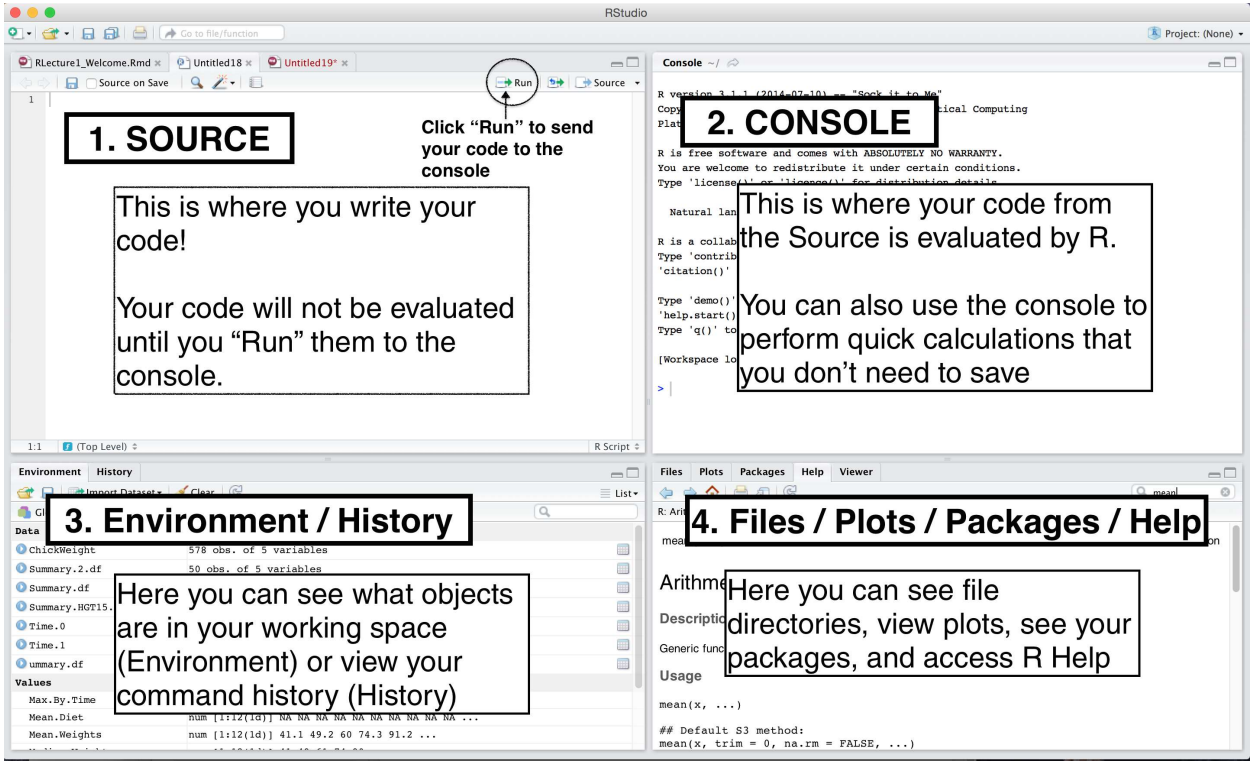
- All analyses will use the R computing language
- Assignments are released after Lab and are due before next lab session
- The instructor/TA will post an answer key to the course website on the due date. For this reason, **late homework will not be accepted**
- Use the homework template to write your answers and submit a **pdf** version on Canvas. Paste your code when required
- Office hours on Thursday 3-5PM at Young Hall 266
- Questions via email at mmcarmo@ucdavis.edu

# Credits

The lab and homework materials heavily rely on the work of previous 103B TAs, Paprika Jiang and Simran Johal.

# R Basics

# The RStudio environment




Source: YaRrr! The Pirate's Guide to R

# R as a calculator

## Common mathematic operations in R:

- `+` : addition
- `-` : subtraction
- `/` : division
- `*` : multiplication
- `^` : exponent
- `sqrt()` : square root
- `exp()` : exponential function

# R as a calculator

```
2 + 2
```

```
[1] 4
```

```
3 - 1
```

```
[1] 2
```

```
2 * 6
```

```
[1] 12
```

```
4 / 2
```

```
[1] 2
```

```
sqrt(16)
```

```
[1] 4
```

# R as a calculator

- To run a line of code just use CTRL + ENTER (that's COMMAND + ENTER if you're on a Mac)
- How would I ask R to divide 10 by 2?

```
1 10 / 2
```

```
[1] 5
```

- How would I ask R to calculate 5 to the 4th power?

```
1 5^4
```

```
[1] 625
```

# R as a calculator

The R Calculator follows the PEMDAS rule:

Parentheseis, exponents, multiplication, division, addition, subraction from left to right

```
3 + 4 * 12
```

```
[1] 51
```

```
(3 + 4) * 12
```

```
[1] 84
```

# Commenting

- Anything that you type after the "#" will be disregarded by R
- You can make notes to yourself and it will not interfere with your code

```
1 # This is a comment to my code
2 1 + 1
```

```
[1] 2
```

# Creating Objects

- We use objects to store any kind of information in R
- Think of an object as a label for a piece of information
- We can save information as variables by using the assignment operator: `<-` or `=`
- General pattern:

```
1 a <- 2 + 2
```

- Now we can access this piece of information by 'calling' `a`

```
1 a
```

```
[1] 4
```

# Creating Objects

- R is case sensitive

```
A
# Error: object 'A' not found
```

- Objects can be overwritten

```
a <- 3
a
```

```
[1] 3
```

# Labeling conventions

Bad

```
1a <- 3
!a <- 3
a! <- 3
```

Better

```
a1 <- 3
a_object <- 3
a.object <- 3
aObject <- 3
```

# Data Types

# Numeric

```
1 class(18.9)
```

```
[1] "numeric"
```

```
1 my_number <- 13
2 class(my_number)
```

```
[1] "numeric"
```

```
1 is.numeric(my_number)
```

```
[1] TRUE
```

# Character

Character data types must be surrounded by quotation marks

```
"student"
```

```
[1] "student"
```

```
myFirstCharacter <- "myFirstCharacter"
myFirstCharacter
```

```
[1] "myFirstCharacter"
```

- What happens if you don't surround a character data type with quotation marks?

```
student
# Error: object 'student' not found
```

- Without quotation marks, R thinks this is an object label.

# Logical

- There are only two options TRUE and FALSE
- They must be in all caps

```
1 true <- FALSE
2 true == TRUE
```

```
[1] FALSE
```

- You could also abbreviate TRUE to T and FALSE to F (not recommended)

# Vectors

- Vectors allow saving multiple pieces of information to an object
- The individual values within a vector are called “elements”
- To do this we can use the `c()` function
- This function combines different pieces of information together

```
c(1, 2, 3, 4)
```

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

```
(1, 2, 3, 4)
# Error: unexpected ',' in "(1,"
```

# Vectors

- We can save these pieces of information to an object

```
1 first_vector <- c(1, 2, 3, 4)
```

- We can look at this object by calling it through the object label

```
1 first_vector
```

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

- Can you guess the class of `first_vector`?

```
1 class(first_vector)
```

```
[1] "numeric"
```

# Vectors

- We saved four pieces of information (four numbers) to the vector
- We can check the number of elements in a vector with the `lenght()` function

```
1 length(first_vector)
```

```
[1] 4
```

- A vector is a **one-dimensional** collection of information
- All the elements must be the **same type**

```
1 second_vector <- c(1, 2, 3, "four")
```

```
1 second_vector
```

```
[1] "1"    "2"    "3"    "four"
```

```
1 class(second_vector)
```

```
[1] "character"
```

# Subsetting vectors

- To pull out one element from a vector that has multiple elements, we need to subset the vector
- Use square brackets `[]` after the label name with the element number we would like to recover inside the brackets
- `myvector[1]`

# Subsetting vectors

- What if I want to pull out the third element from a vector?

```
second_vector[3]
```

```
[1] "3"
```

- We can save this as a new object

```
third_element <- second_vector[3]
third_element
```

```
[1] "3"
```

- We can also pull out multiple elements at once

```
second_vector[c(2, 3, 4)]
```

```
[1] "2"    "3"    "four"
```

# Functions in R

- Functions are pre-written pieces of code that accomplish some task
- Rather than writing out the code to do this task, we can call a function by its label and it will complete that task
- Let's say I wanted to calculate the mean (or average) of our numeric vector

```
1  (1 + 2 + 3 + 4) / 4
```

```
[1] 2.5
```

# Functions in R

- What if `first_vector` had 100 numbers not 4?
- Or what if you had 100 vectors and wanted to calculate the means of each one
- We could use the `sum()` function and the `length()` function
  - `sum()` will add up all the numbers in a vector
  - `length()` tells you how many elements are in a vector

```
1 sum(first_vector)/length(first_vector)
```

```
[1] 2.5
```

- Or we could use the `mean()` function

```
1 mean(first_vector)
```

```
[1] 2.5
```

# Functions in R

## Arguments

- Arguments are the information we give the function so it can carry out its task
- A function can have multiple arguments
- `functionLabel(argument1, argument2, argument3)`
- For the `mean()` function, the first argument was the data or the number we wanted the mean of

# Functions in R

- `round()` will round whatever number you give in the first argument

```
round(3.666)
```

```
[1] 4
```

- What if I want it to round the number to the second decimal point?
- We can add another argument

```
round(3.666, 2)
```

```
[1] 3.67
```

```
# The second argument tells it how many decimal points to round to
```

# Functions in R

- Each argument has a label
- Sometimes we don't use them for convenience, but it is helpful specially if you're dealing with a function with many arguments

```
round(x = 3.666, digits = 2)
```

```
[1] 3.67
```

- If you use labels, the order of the arguments don't matter

```
round(digits = 2, x = 3.666)
```

```
[1] 3.67
```

# Functions in R

- What happens if we write the digits argument first without the labels?

```
round(2, 3.666)
```

```
[1] 2
```

- When you don't use labels, the order **really** matters
- To learn more about a pre-built function and its arguments, type `?function_name()` in the console (e.g., `?round()`) to find the help page of that function
- Or you can press tab once your cursor is within the function parenthesis

# Functions in R

- Another useful function is `class()`
- The class function will tell you what kind of data type

```
1 class(first_vector)
```

```
[1] "numeric"
```

```
1 class(second_vector)
```

```
[1] "character"
```

# Data Structures

# Types of data structure

1- Vector

2- Matrix

3- Array

4- Data frame

5- List

# Matrix

- Two-dimensional dataset (has columns and rows) of **one data type**
- `matrix(data, nrow, ncol, byrow)`

```
m <- matrix(data = c(1:6), # The matrix will contain elements 1, 2, 3, 4, 5, 6
            nrow = 2, # this matrix will have two rows
            ncol = 3, # this matrix will have three columns
            byrow = TRUE) # we fill the elements by the row
m
```

```
     [,1] [,2] [,3]
[1,]    1    2    3
[2,]    4    5    6
```

- What happens if you change `byrow = FALSE` ?

# Matrix

- `dim()` tells you the dimension of a matrix
- First element is number of rows, second element is number of columns

```
1 dim(m)
```

```
[1] 2 3
```

# Matrix

- We can also create a matrix by combining vectors

```
vec1 <- 1:5
vec2 <- 6:10
```

- We can use the `cbind()` function to make each vector their own column in a matrix

```
col_matrix <- cbind(vec1, vec2)
col_matrix
```

```
     vec1 vec2
[1,]    1    6
[2,]    2    7
[3,]    3    8
[4,]    4    9
[5,]    5   10
```

# Matrix

- Or we can use the `rbind()` function to make each vector their own row in a matrix

```
row_matrix <- rbind(vec1, vec2)
row_matrix
```

```
     [,1] [,2] [,3] [,4] [,5]
vec1    1    2    3    4    5
vec2    6    7    8    9   10
```

# Data frames

- Allow you to have **multiple** data types
- We can use the `data.frame()` function to create a data frame
- Each argument is a different column
- We can also add labels to each column

```r
dat <-  data.frame(
  schoolyear = c("Freshman", "Sophomore","Freshman"),
  height = c(152, 171.5, 165),
  weight = c(81,93, 78),
  age = c(18,20,19)
)
```

# Subsetting Two-Dimensional Objects

- We've used square brackets `[]` to subset vectors
- We can use square brackets `[]` to subset two-dimensional objects like a matrix or data frame
- `twoDimObject[row#, column#]`

# Subsetting Two-Dimensional Objects

- If I want to subset the element in row 3 and column 2

```
dat[3, 2]
```

```
[1] 165
```

- If I want to subset the entire 3rd column

```
dat[, 3]
```

```
[1] 81 93 78
```

# Subsetting Two-Dimensional Objects

- I could also subset by column name

```
dat[, "weight"] # this subsets all the rows from the weight column
```

```
[1] 81 93 78
```

```
dat[1, "weight"] # this subsets the first row from the weight column
```

```
[1] 81
```

- If we just want to subset a column we can use the `$` operator: `twoDimObject$ColumnName`

```
dat$schoolyear
```

```
[1] "Freshman"  "Sophomore" "Freshman"
```

# Review of Statistical Concepts

PSC 103B - Statistical Analysis of Psychological Data

UCDAVIS

# Central tendency: Mean, Median, and Mode

- These are the 3 **most common** measures of central tendency
- They are used to **describe a distribution of observations** (e.g., all the grades on an exam) in one number that best represents that distribution
- Suppose we asked a bunch of UC Davis students how many hours per week they spent watching Netflix, and how many hours they spent exercising during Winter break:

```
1 netflix <- c(2, 6, 1, 7, 2, 4, 11, 40, 7, 0, 3, 4, 5, 2, 15)
2 exercise <- c(2, 2, 6, 2, 12, 45, 8, 3, 2, 6, 4, 0, 1, 3, 0)
```

# Central tendency: Mean, Median, and Mode

- How many observations are in each variable?

```
1 length(netflix)
```

```
[1] 15
```

```
1 length(exercise)
```

```
[1] 15
```

- Let's take a look at the average time each student spent on these activities:

```
1 mean(netflix, na.rm = TRUE) # use the argument na.rm = TRUE to ignore missing values
```

```
[1] 7.266667
```

```
1 mean(exercise, na.rm = TRUE)
```

```
[1] 6.4
```

- Unsurprisingly, students exercised less than they watched netflix, on average.

# Central tendency: Mean, Median, and Mode

- But is the mean a good representation of these data?
- Take a look again at the values and see if you find something odd

```
1 netflix
```

```
[1]  2  6  1  7  2  4 11 40  7  0  3  4  5  2 15
```

```
1 exercise
```

```
[1]  2  2  6  2 12 45  8  3  2  6  4  0  1  3  0
```

- One person is watching Netflix 40h a week
- Another exercised 45h per week

# Central tendency: Mean, Median, and Mode

- When we have **outliers**, sometimes the **median** is a better representation of the data
- Remember, the median is the **middle value** of your data, after you have ordered it

```
1 median(exercise, na.rm = TRUE)
```

```
[1] 3
```

```
1 median(netflix, na.rm = TRUE)
```

```
[1] 4
```

# Central tendency: Mean, Median, and Mode

- Sometimes, we can't do arithmetic on the data we have
- If we had asked our 15 participants what their favorite flavor of ice cream was, we would not be able to describe that distribution using a mean or a median. We would have to use the **mode**
- The mode is just the most frequent value

# Central tendency: Mean, Median, and Mode

- R doesn't have a function for because it is not very frequently used, so we use the `table()` function
- `table()` gives you the number of times each element shows up in an object

```
1 table(netflix)
```

```
netflix
 0  1  2  3  4  5  6  7 11 15 40
 1  1  3  1  2  1  1  2  1  1  1
```

- You can use the `sort()` function on the result of the table function to order it

```
1 sort(table(exercise))
```

```
exercise
 1  4  8 12 45  0  3  6  2
 1  1  1  1  1  2  2  2  4
```

# Spread: Variance and Standard Deviation

- Do all the students exercise about the same? Or do some students exercise a lot while others don't?
- How are the observations spread out around the mean or median?
- We're gonna look at two different kinds that are related: **Variance** and **standard deviation**

# Spread: Variance

## 1. Calculate the mean

```
1 mean(exercise, na.rm = TRUE)
```

```
[1] 6.4
```

## 2. Find the distance from each observation to the mean

```
1 exercise - mean(exercise, na.rm = TRUE)
```

```
 [1] -4.4 -4.4 -0.4 -4.4  5.6 38.6  1.6 -3.4 -4.4 -0.4 -2.4 -6.4 -5.4 -3.4 -6.4
```

- Let's save that!

```
1 diffs <- exercise - mean(exercise, na.rm = TRUE)
```

# Spread: Variance

## 3. Square the differences

```
1 diffs_sq <- diffs^2
```

## 4. Sum everything and divide by N-1

```
1 sum(diffs_sq)/(length(exercise) - 1)
```

```
[1] 124.4
```

- We can check our answers using `var()`

```
1 var(exercise, na.rm = TRUE)
```

```
[1] 124.4
```

# Spread: Standard deviation

- To get the **standard deviation**, we just get the square root of the variance:

```
ex_var <- var(exercise, na.rm = TRUE)
sqrt(ex_var)
```

```
[1] 11.15347
```

- Or, R has a `sd()` function:

```
sd(exercise, na.rm = TRUE)
```

```
[1] 11.15347
```

- We often prefer to use the standard deviation, because its units are the same units of our variable

# Relationships between variables: correlation and covariance

- In our imaginary example, each person gave us two bits of information, exercise and netflix hours
- Let's organize our data into a dataframe to better keep track of it:

```r
df <- data.frame(Netflix = netflix,
                 Exercise = exercise,
                 stringsAsFactors = FALSE)
df
```

```
  Netflix Exercise
1       2        2
2       6        2
3       1        6
4       7        2
5       2       12
6       4       45
7      11        8
```

```
8         40         3
9          7         2
10         0         6
11         3         4
12         4         0
13         5         1
14         2         3
```

If you are in RStudio, you can look at df by clicking on it, using `View()`, typing it in the console or using functions like `head()` and `tail()`. Can you tell what those do?

```
1 head(df)
```

```
  Netflix Exercise
1       2        2
2       6        2
3       1        6
4       7        2
5       2       12
6       4       45
```

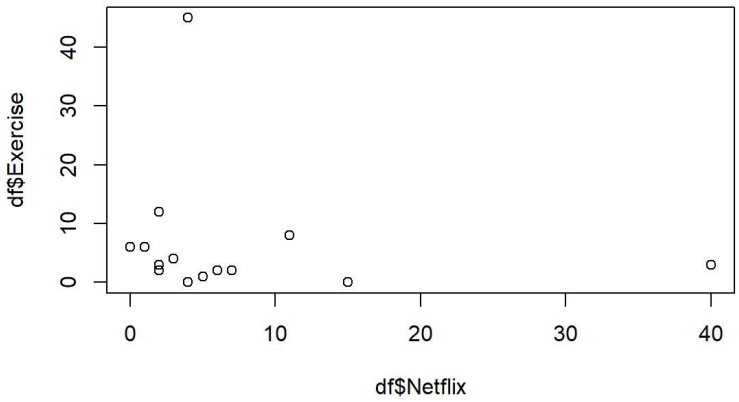
```
1 tail(df)
```

```
   Netflix Exercise
10       0        6
11       3        4
12       4        0
13       5        1
14       2        3
15      15        0
```

## It's also a good idea to plot your data

- This helps you get a general idea for what your data looks like, and to see if there is anything weird going on
- To make a scatterplot, we can use `plot()` function in R

```
1 plot(x = df$Netflix, y = df$Exercise)
```


df$Exercise
0
10
20
30
40
df$Netflix
0
10
20
30
40

# We can make our plot look prettier changing the axis labels, and even giving our plot a title

```
plot(df$Netflix, df$Exercise, xlab = "Hours Spent Watching Netflix", #changes the x-axis label
     ylab = "Hours Spent Exercising", # changes the y-axis label
     main = "Plot of Time Spent Watching Netflix vs. Exercising over Break") # gives your plot a tit
```

# Covariance

- What if we wanted to quantify this relation? We can use the covariance and correlation!
- The **covariance** between two variables is a measure of how the two variables change together
- It only makes sense if there's some connection between the two variables
- It resembles the variance, but instead of squared differences from the mean, we multiply these differences from the mean by each other

# Covariance

## 1. Get the differences from the mean for each variable:

```
diff_nfx <- df$Netflix - mean(df$Netflix, na.rm = TRUE)
diff_ex <- df$Exercise - mean(df$Exercise, na.rm = TRUE)
# you can check this is right -- the differences will sum to 0 (or very close to it)
sum(diff_ex)
```

```
[1] -3.552714e-15
```

## 2. Multiply them by each other

```
mult_diffs <- diff_nfx * diff_ex
```

# Covariance

## 3. Sum all these multiplied differences

```
1 sum_diffs <- sum(mult_diffs, na.rm = TRUE)
```

## 4. Divide by N - 1

```
1 cov_NetEx <- sum_diffs/14
2 cov_NetEx
```

```
[1] -15.18571
```

- We see that the covariance is **negative**, indicating that the Netflix and Exercise variables are **inversely related** to each other

# Covariance

- You can verify this yourself by using the cov() function

```
1 cov_NetEx_2 <- cov(df$Netflix, df$Exercise, use = "complete.obs")
2 cov_NetEx_2
```

```
[1] -15.18571
```

- For `cov()`, the `use = "complete.obs"` argument acts similarly to `na.rm = TRUE`: it will only use data from people who gave an answer to both Netflix and exercise

# Correlation

- We don’t know how strong is this association because covariances have **arbitrary** scales based on the scales of the original variables
- We don’t know how big they could get so we don’t know if this value is large or small
- **Correlations** can only range between **-1** and **1**, so they’re easier to interpret

# Correlation

- We'll standardize the covariance to get a correlation
- Standardizing in this case means dividing by the variables' standard deviations:

```
sd_N <- sd(df$Netflix, na.rm = TRUE)
sd_E <- sd(df$Exercise, na.rm = TRUE)
```

- We divide the covariance by the product of the variances to get a correlation:

```
cov_NetEx/(sd_N*sd_E)
```

```
[1] -0.1377893
```

- And we can check this using the cor() function:

```
cor(df$Netflix, df$Exercise, use = "complete.obs")
```

```
[1] -0.1377893
```

```r
cor(df$Netflix, df$Exercise, use = "complete.obs")
```

```
[1] -0.1377893
```

Since the correlation ranges between -1 and 1, we can say something about the strength of this relation

- Based on some rules of thumb we can say there is a weak negative relation between watching Netflix and exercise
- What is considered strong vs. weak can **depend on the area of research** you're in
- Next week: statistical test to see whether this correlation is significantly different from 0 or not!

PSC 103B - Statistical Analysis of Psychological Data

UC DAVIS