LeaRning Week 1

LeaRning Team

4/20/2021

Introduction

```
##
##  --------------
## Welcome to LeaRning!
##  --------------
##     \
##       \
##         \
##               /""-._
##               .       '-,
##                :          '',
##                 ;      *     '.
##                  ' *         () '.
##                    \               \
##                     \      _.---.._ '.
##                     :  .' _.--''-''  \ ,'
##         ..-_          '/.'             . ;
##         ; `-.          ,                \'
##          ;   `,         ;              ._\
##           ;    \     _,-'                 ''--._
##           :    \_,-'                          '-._
##           \ ,-'                      .            '-._
##           .'          __.-'';           \...,__        '.
##         .'      _,-'        \             \   ''--.,__  '\
##         /    _,--' ;         \              ;           \^.}
##         ;_,-' )     \  )\      )            ;
##              /       \/  \_.,-'             ;
##             /                              ;
##           ,-'  _,-'''-.    ,-.,            ;      PFA
##       ,-' _.-'         \  /    |/'-._...--'
##      :--``              )/
##   '
##
```

Hi there! Welcome to the world of R. This initial lesson is a gentle introduction to R objects and how to navigate them. We will cover the basics of characters, factors, and numbers, and how they are organized into vectors, data frames, and lists. Finally, we will put R to use by already delving into statistics with a t-test!

Let's start with basic R objects.

Part 1: Basic Objects

In R, objects appearing in your environment are used to accomplish tasks. Anything you write in your script will be taken as instructions for R to do something. So let's start with something simple: non-executable text.

```
# This is a note to myself
# :)
```

Any text that follows a # is just writing. You can use this to write notes to yourself, annotate your codes, create titles, etc. However, ANYTHING you type beyond this reflects real-life coding. So let's get to it.
There are three basic types of objects:
1. Characters
2. Factors
3. Numbers or Integers

1.1 Characters

Characters are just text. Let's make one.

```
"Four"
```

```
## [1] "Four"
```

A note on creating objects

What happened? What if we want to save an object? We can create objects and save them by assigning them to a representative value. This is done using <- or =

```
x <- "Four"
x
```

```
## [1] "Four"
```

```
# same thing

x = "Four"
```

Have a look at your environment. Congratulations! You have created your first object.

NOTE: executing code

You can execute code by placing your cursor anywhere on that line and hitting command+enter
You can also execute a selection of code by highlighting it
If you highlight a part of a line, R will only execute what is highlighted.
Now let's try one more thing.

```
x <- Four
```

```
## Error in eval(expr, envir, enclos): object 'Four' not found
```

Didn't work! This is an R-ism. Anything that isn't a number must be in your environment in order to be assigned a value. Let's re-assign x.

```
x <- "Four"

y <- x # you can assign objects values of other objects that already exist too.
```

Let's do some math

```
y <- "Five"

x + y
```

```
## Error in x + y: non-numeric argument to binary operator
```

How might we want to do that instead? With numbers!

1.2 Numbers

Unlike letters, numbers typed in R are just taken as numbers. Phew! something intuitive.

```
x <- 4
y <- 5
x + y
```

```
## [1] 9
```

A word of caution:

Watch out - it is easy to overwrite your objects

```
x <- 6
```

now x is six. There is no going back. If you want x to be four again you have to re-assign it!!!!

Now let's cover the last class: Factors!

1.3 Factors

Factors in R represent categorical data. These are finite integer values that are represented by characters, making it easier for us to differentiate them.

```
pets <- c("cat", "dog", "ostrich")
pets <- factor(pets)
pets
```

```
## [1] cat     dog     ostrich
## Levels: cat dog ostrich
```

Factors are "labelled numbers". Imagine, for regression, that you have "Males" and "Females". Regression reads these as 0 and 1. To make it easier for you, you can label these. These are called "dummy labels".

A note on Functions

Notice above that we used the ( ). Recall that this follows a function. Above, the "c" (for concatenate) function took the characters "male" and "female" and turned it into a vector. We will use a few basic functions below. Watch out for them... they will each be followed by ( )

Know your classes

To be sure, you can always find out the class of your object using the class() function

```
class(x)
```

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

```
class(y)
```

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

```
class(pets)
```

```
## [1] "factor"
```

```
name <- "Nelly"
class(name)
```

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

Now that we know our objects, let's have a look at how they're organized.

Part 2: Vectors, Data Frames, and Lists

2.1. Vectors

A vector is a string of objects. These can be characters, factors, or numbers. Let's make some vectors.

```
my.first.vector <- c(x,y)
my.first.vector
```

```
## [1] 6 5
```

```
another.vector <- c(1,2,3,4,5)
another.vector
```

```
## [1] 1 2 3 4 5
```

```
yet.another.vector <- c(1:40)
yet.another.vector
```

```
##  [1]  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23
## [24] 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40
```

```
and.one.more <- c("red", "blue", "green")
and.one.more
```

```
## [1] "red"   "blue"  "green"
```

Just like objects, vectors fall into classes.

```
class(my.first.vector) # numbers
```

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

```
class(another.vector) # more numbers
```

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

```
class(yet.another.vector) # integer
```

```
## [1] "integer"
```

```
class(and.one.more)  # characters
```

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

```
class(pets) # factors
```

```
## [1] "factor"
```

But what about mixing vector classes?

```
mix.it.up <- c(another.vector, and.one.more)
mix.it.up
```

```
## [1] "1"     "2"     "3"     "4"     "5"     "red"   "blue"  "green"
```

Vectors cannot contain mixed classes. They can either be all numbers, all characters, or all factors. Be careful when combining different kinds of information... best way is to ALWAYS check your output.

NOTE: Converting classes

It is simple to get from one class to another. We use the "as" family of functions to go from one class to another. These "as" functions include, among others,
as.character()

as.numeric()
as.factor()

They are intuitive, but have a few subtle rules of their own. Let's explore.

```
# Turn numbers into characters
wordy.vector <- as.character(another.vector)
wordy.vector
```

```
## [1] "1" "2" "3" "4" "5"
```

you can't quite go from words to numbers though

```
# Trying to turn characters into numbers
colors <- as.numeric(and.one.more)
```

```
## Warning: NAs introduced by coercion
```

```
colors
```

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

A note on NAs

As you can see, we get a warning message. R automatically replaced the characters by NA in the above code.

NA is the 'value' attributed to emptiness in R. That is, if a variable is declared but empty, it will hold the 'value' NA.
Usually if R cannot process something, like trying to convert words into numbers, it will assign NA to it.

you CAN though go from character-words back to numnbers though.

```
# going from word-numbers back to numbers
numbers.2 <- as.numeric(wordy.vector)
numbers.2
```

```
## [1] 1 2 3 4 5
```

Let's revisit the factors. Remember that they are labelled integers? If you convert a factor to an integer, the character labels are dropped, and we see the numbers that were hidden underneath.

```
# Converting a factor to integers
pets2 <- as.integer(pets)
pets2
```

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

2.2 Lists

Lists are ways to put varous different objects together. Think a folder on your computer.

```
my.list <- list(my.first.vector, another.vector, yet.another.vector)
my.list
```

```
## [[1]]
## [1] 6 5
##
## [[2]]
## [1] 1 2 3 4 5
##
## [[3]]
##  [1]  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23
```

```
## [24] 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40
```

2.3 Data Frames

Now we are back in familiar territory. Data frames are essenitally what most of use when we work with in excel. In R, they need to stay in a tidy format, but more on that later. For now let's just make a data frame.

```
animals <- c("cat", "dog", "ostrich")
pet <- c("Y", "Y", "N")
size <- c(5, 8, 20)

my.data.frame <- data.frame(animals, pet, size)
my.data.frame
```

```
##   animals pet size
## 1     cat   Y    5
## 2     dog   Y    8
## 3 ostrich   N   20
```

You can view your data frame by clicking on it in your environment. You can also view it using the View function.

```
View(my.data.frame)
```

Part 3: Navigating objects in R

3.1 Navigating Vectors

Remember from our lesson, that we navigate objects by placing a square bracket after the object name. For vectors, the value represents the nth term in your vector (vector[3] for the third item). For data frames, this is [row,column] (example: my.data.frame[row,column]). Now let's give these a try.

```
# navigate a vector
another.vector[3]
```

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

```
#OR

another.vector[1:2]
```

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

QUESTION: How would you code for the first and the third item in your vector?

```
another.vector[1,3]
```

```
## Error in another.vector[1, 3]: incorrect number of dimensions
```

Any ideas why this didn't work? R interpreted the , as separating different types of coordinates (think row vs. columns). If you are navigating a vector and you want to select multiple values that aren't sequential (i.e. a simple x:y won't do it), then you need to concatenate the values.

QUESTION: who remembers the function for concatenating? Hint - we've just used it above

```
another.vector[c(1,3)]
```

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

3.2 Navigating lists

Lists get a bit tricky. You can navigate it in two main ways:
1. Get a segment of the list. The result is a list, subset to what you selected.
2. Pull out an object from the list. The object is no longer a part of a list.

These are differentiated by placing the value coordinates in one vs. two sets of square brackets

```
# Example 1: I want to subset the list to the first two items
short.list <- my.list[c(1:2)]

# Example 2: I want to get the first item out of my list, and save it as an object
x <- my.list[[1]]
```

As you can see, if you have long lists, it may not be intuitive to navigate it using coordinates. Thankfully, there is a better way.
You can name your list for easier navigation.

```
names(my.list) <- c("Nelly", "Adrien", "Emma")
my.list$Nelly
```

```
## [1] 6 5
```

In R, the $ sign is generally used to intuitively navigate named objects, such as lists, and as we'll later see, data frames too.

3.3 Navigating Data Frames

Recall, that for a data frame you can get entire rows, entire columns, or specific values in a row-column coordinate.

Also remember: navigating a data frame is ALWAYS [row,column]

```
# Example: I want the first row of my data frame
my.data.frame[1,]
```

```
##   animals pet size
## 1     cat   Y    5
```

QUESTION: How would you get the first column from my data frame?

```
my.data.frame[,1]
```

```
## [1] cat     dog     ostrich
## Levels: cat dog ostrich
```

QUESTION: How would you get the value in the first row and second column?

```
my.data.frame[1,2]
```

```
## [1] Y
## Levels: N Y
```

QUESTION: How can you get a few columns (think back to what we needed to to do get multiple items from a vector)

```
my.data.frame[,c(1,2)]
```

```
##   animals pet
## 1     cat   Y
## 2     dog   Y
## 3 ostrich   N
```

```
# You can also achieve this by substituting column names for numerical coordinates.
my.data.frame[,c("animals","pet")]
```

```
##   animals pet
## 1     cat   Y
## 2     dog   Y
## 3 ostrich   N
```

Again, navigating a data frame can be cumbersome if you always have to remember the coordinates in all of your columns. How much easier would it be if R had a way of letting you use the column names instead? Well, thankfully there is a way with $.

```
# getting a column: the coordinate way
animals <- my.data.frame[,1]
animals
```

```
## [1] cat     dog     ostrich
## Levels: cat dog ostrich
```

```
# Now try this
animals <- my.data.frame$animals
animals
```

```
## [1] cat     dog     ostrich
## Levels: cat dog ostrich
```

NOTE: If you hit "tab" after your dollar sign, R nicely prints all the names for you. Thanks R!

Part 4: Putting R to real use

Enough of this Syntax! I want to put R to some use now. So, before we close, let's use some of the skills we learned to perform a useful function - the T test!
First, let's find out more about this function.

```
?t.test()
```

Note: In R, if you want to learn more about the useage of any function place a "?" in front of the function name and the help file will come up on your viewer.

Based on our help file, the t.test function takes two numeric vectors that you want to compare. By specifying the alternative, you can choose between a one-sided or two-sided t-test. Let's do a two-sided.

```
# first some data
cat.weights <- c(15, 12, 9, 11, 9)
dog.weights <- c(20, 25, 21, 23, 22)

# Now, the test
t.test(cat.weights, dog.weights, alternative = "two.sided")
```

```
##
##  Welch Two Sample t-test
##
## data:  cat.weights and dog.weights
## t = -7.8174, df = 7.5204, p-value = 7.216e-05
## alternative hypothesis: true difference in means is not equal to 0
## 95 percent confidence interval:
##  -14.281208  -7.718792
## sample estimates:
```

```
## mean of x mean of y
##      11.2      22.2
```

That's great! Now - what if we want to use our result later or save it? We will have to save the result to an object.

```
result <- t.test(cat.weights, dog.weights, alternative = "two.sided")
class(result)
```

```
## [1] "htest"
```

The "htest" class denotes a hypothesis test result. Any statistical test result in base R will carry this class (e.g. Kruskal-Wallis, ANOVA, etc..) However, let's take a look at the "result" object in our environment. It looks a lot like a named list.

QUESTION: How would you want to get the p-value out of the list?

```
result$p.value
```

```
## [1] 7.215823e-05
```

QUESTION: How about saving the p-value, so you can use it later?

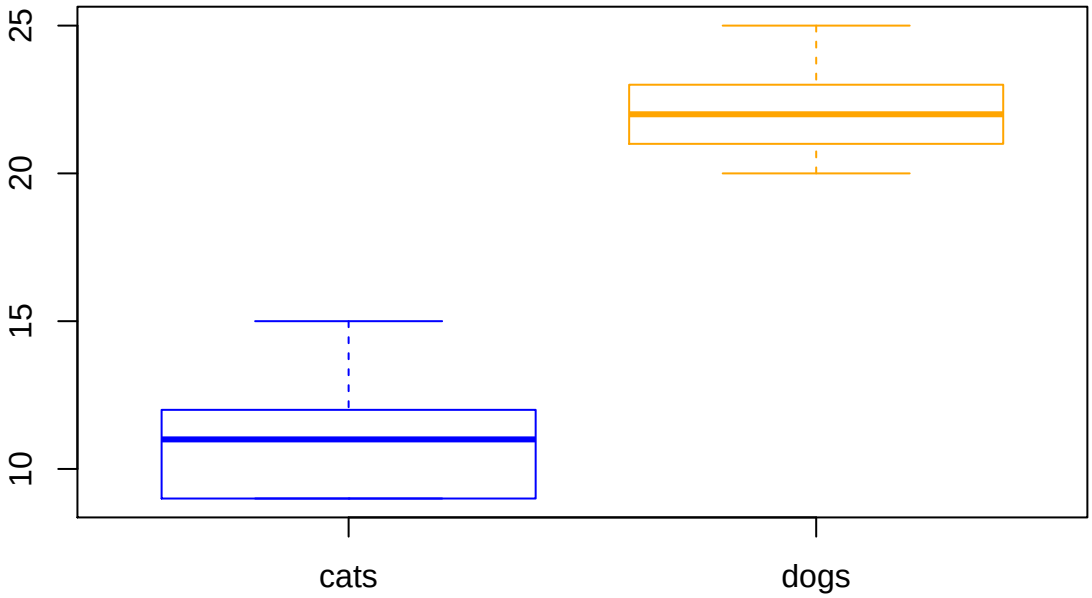
```
pval <- result$p.value
pval
```

```
## [1] 7.215823e-05
```

Bonus

Let's finish off with a quick plot of our results. We will tackle plotting in much more depth in the weeks to come, but what's a result without a plot? For now, let's make a quick boxplot in base R with the "boxplot" function.

```
boxplot(cat.weights, # specify the first vector to plot
        dog.weights, # specify the second vector to plot
        names = c("cats", "dogs"), # specify the x-axis labels
        border = c("blue", "orange")) # specify the border colors
```



And that's it for today! Congratulations. You can now:

1. Familiar with the R studio interpase
2. Understand the basic types of objects often used in R are characters, numbers, and factors
3. Know that these objects are futher organized in vectors, lists, and data frames
4. Can perform basic navigation of these objects and
5. Have an example of how numeric vectors are used to perform useful functions in R.

Next class is all about data frames! We will out our navigation sets to use by reading a data set into R, navigating the type of information it contains, and manipulate the data set to test a hypothesis.

```
##
##  --------------
## See you next time!
##  --------------
##     \
##       \
##         \
##              |\___/|
##           ==) ^Y^ (==
##             \  ^  /
##              )=*=(
##             /     \
##             |     |
##            /| | | |\
##            \| | |_|/\
##       jgs  //_// ___/
##                \_)
##
```