HLC R Class - Day 1

Elena Kharitonova

6/6/2022

Welcome to HLC: Introduction to R

- Introductions
- Class expectations:
  - Work together
  - Ask questions
  - Be respectful of others
  - Making mistakes is the best way to learn!
  - Homework Policy
    - We will assign homework but we won't grade it. We plan to hand out solutions after it is due. If anybody has questions about any of the problems, we can go over them in the first 10 minutes of class.

R setup

- Download R and Rstudio
- Open Rstudio
- Explain each window

What is R and Rstudio? How are they different?

R is a programming language used for statistical computing. It is the software that performs the actual instructions. Without R installed on your computer or server, you would not be able to run any commands. Meanwhile, RStudio is an IDE, or integrated development environment. It's a software that provides an nifty interface to R, that makes using the R language a lot easier and neater. RStudio allows users to develop and edit programs in R by supporting a large number of statistical packages, higher quality graphics, and the ability to manage your workspace.

R and RStudio are not separate versions of the same program, and cannot be substituted for one another. R may be used without RStudio, but RStudio may not be used without R.

Console vs. R scripts vs. R markdown

There are two main ways of interacting with R: using the console or using a file that contains your code.

The console window (in RStudio, the bottom left panel) is the place where R is waiting for you to tell it what to do, and where it will show the results of a command. You can type commands directly into the console, but they will be forgotten when you close the session. If R is ready to accept commands, the R console shows a > prompt. If it receives a command, R will try to execute it, and when ready, show the results and come back with a new >-prompt to wait for new commands. (We recommend using the console only for things you don't want to keep a record of)

If R is still waiting for you to enter more data because it isn't complete yet, the console will show a + prompt. It means that you haven't finished entering a complete command. This is because you have not 'closed' a

parenthesis or quotation. If you're in RStudio and this happens, click inside the console window and press Esc; this should help you out of trouble.

An R script file (.R file) allows you to enter the commands and save the script. (It is the top left window in RStudio) This way, you have a complete record of what you did, you can easily show others how you did it and you can do it again later on if needed. You can evaluate any line of code a by using Ctrl-Enter or by highlighting it and clicking Run.

An R Markdown (.Rmd file) is a specific type of file format for making dynamic documents, like this one. It is nice because it allows you to write plain text and code in one document. Therefore in one document you can save and execute code, and produce high quality documents that include both code and text. Additionally, R Markdown files can be converted to an .html, a .pdf, or a .word document easily by knitting.

For the purposes of our instruction, R Markdown allows us to include code chunks and the text that helps explain them in an easy-to-read manner. For your own use, R Markdown will allow you to write documents and reports that include traditional formatted text, as well as the data visualizations you make in class, and present them both together in a high quality professional document.

Using this document

- Code blocks and R code have a grey background
- # indicates a comment, and anything after a comment will not be evaluated in R
- The comments beginning with ## under the code in the grey code boxes are the output from the code directly above; any comments added by us will start with a single #
- While you can copy and paste code into R, you will learn faster if you type out the commands yourself.
- Read through the document after class. This is meant to be a reference, and ideally, you should be able to understand every line of code. If there is something you do not understand please email us with questions or ask in the following class (you're probably not the only one with the same question!).

Intro to R

By default R spits out things to terminal

```
2 + 2
```

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

```
100*100
```

```
## [1] 10000
```

```
"My name is Spencer"
```

```
## [1] "My name is Spencer"
```

You can store things in variables with <-

You can quickly type `<-` with `Alt + -` in Rstudio

```
myName <- "Spencer"
```

R will spit it back out when you call it

```
myName
```

```
## [1] "Spencer"
```

R can do operations on variables

```
myNumber <- 10

myNumber + 10
```

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

Comments are hidden from R

```
# Text after a # will not be run, it is a "comment".
# This lets you document your scripts. Always comment your code!
# The # does not have to be at the beginning

2 + 2 # Addition!
```

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

Data in R

Vectors The most simple data structure available in R is a vector. You can make vectors of numeric values, logical values, and character strings using the `c()` function. For example:

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

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

```
c(TRUE, TRUE, FALSE)
```

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

```
c("a", "b", "c")
```

```
## [1] "a" "b" "c"
```

You can also join to vectors using the `c()` function.

```
x <- c(1, 2, 5)
y <- c(3, 4, 6)
z <- c(x, y)
z
```

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

An element of a vector can be selected using `[]`. The `[` operator can also take a vector as the argument. For example you can select the first and third elements:

```
z[3]
```

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

```
z[c(3,1)]
```

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

Data frames Data frames are likely the data structure you will used most in your analyses. A data frame is a special kind of list that stores same-length vectors of different classes. You create data frames using the `data.frame` function. The example below shows this by combining a numeric and a character vector into a data frame. It uses the `:` operator, which will create a vector containing all integers from 1 to 3.

```r
df1 <- data.frame(x = 1:3, y = c("a", "b", "c"))
df1
```

```
##   x y
## 1 1 a
## 2 2 b
## 3 3 c
```

```r
class(df1)
```

```
## [1] "data.frame"
```

`iris` is a built-in dataset that we will use today to cover some basic. It contains measurements of 3 different species of flower.

We can look at the contents by printing it to the console (annoying for large datasets)

```r
iris
```

The `head()` function will print the first 6 rows

```r
head(iris)
```

```
##   Sepal.Length Sepal.Width Petal.Length Petal.Width Species
## 1          5.1         3.5          1.4         0.2  setosa
## 2          4.9         3.0          1.4         0.2  setosa
## 3          4.7         3.2          1.3         0.2  setosa
## 4          4.6         3.1          1.5         0.2  setosa
## 5          5.0         3.6          1.4         0.2  setosa
## 6          5.4         3.9          1.7         0.4  setosa
```

The `View()` function will open the dataset in Rstudio's data viewer.

```r
View(iris)
```

Datasets can be stored in R as **data.frames**. `iris` is a data.frame. You can think of these like excel tables.

What is special about data.frames is that each column contains data of a certain type. R sees numbers and words as different things. Number columns can be used in math operations, but word columns cannot. **Note:** We will talk in more detail about these data types later

You can view the type of each column with the `str()` (Structure) function. Here you can see that `Sepal.Length` is a number (`num`), but `Species` is a type called a `Factor`.

```r
str(iris)
```

```
## 'data.frame':    150 obs. of  5 variables:
##  $ Sepal.Length: num  5.1 4.9 4.7 4.6 5 5.4 4.6 5 4.4 4.9 ...
##  $ Sepal.Width : num  3.5 3 3.2 3.1 3.6 3.9 3.4 3.4 2.9 3.1 ...
##  $ Petal.Length: num  1.4 1.4 1.3 1.5 1.4 1.7 1.4 1.5 1.4 1.5 ...
##  $ Petal.Width : num  0.2 0.2 0.2 0.2 0.2 0.4 0.3 0.2 0.2 0.1 ...
##  $ Species     : Factor w/ 3 levels "setosa","versicolor",..: 1 1 1 1 1 1 1 1 1 1 ...
```

Individual columns can be referenced in R by using `$` to call a column by name.

```r
iris$Sepal.Length
```

```
##   [1] 5.1 4.9 4.7 4.6 5.0 5.4 4.6 5.0 4.4 4.9 5.4 4.8 4.8 4.3 5.8 5.7 5.4 5.1
##  [19] 5.7 5.1 5.4 5.1 4.6 5.1 4.8 5.0 5.0 5.2 5.2 4.7 4.8 5.4 5.2 5.5 4.9 5.0
##  [37] 5.5 4.9 4.4 5.1 5.0 4.5 4.4 5.0 5.1 4.8 5.1 4.6 5.3 5.0 7.0 6.4 6.9 5.5
##  [55] 6.5 5.7 6.3 4.9 6.6 5.2 5.0 5.9 6.0 6.1 5.6 6.7 5.6 5.8 6.2 5.6 5.9 6.1
##  [73] 6.3 6.1 6.4 6.6 6.8 6.7 6.0 5.7 5.5 5.5 5.8 6.0 5.4 6.0 6.7 6.3 5.6 5.5
```

```
##  [91] 5.5 6.1 5.8 5.0 5.6 5.7 5.7 6.2 5.1 5.7 6.3 5.8 7.1 6.3 6.5 7.6 4.9 7.3
## [109] 6.7 7.2 6.5 6.4 6.8 5.7 5.8 6.4 6.5 7.7 7.7 6.0 6.9 5.6 7.7 6.3 6.7 7.2
## [127] 6.2 6.1 6.4 7.2 7.4 7.9 6.4 6.3 6.1 7.7 6.3 6.4 6.0 6.9 6.7 6.9 5.8 6.8
## [145] 6.7 6.7 6.3 6.5 6.2 5.9
```

Similarly, to a vector so you can subset a data frame using `[]`, but keep in mind data frames have both rows and columns!

```
#dataframe[row, column]

## Since Sepal.Length is the first column
iris[,1]
```

```
##   [1] 5.1 4.9 4.7 4.6 5.0 5.4 4.6 5.0 4.4 4.9 5.4 4.8 4.8 4.3 5.8 5.7 5.4 5.1
##  [19] 5.7 5.1 5.4 5.1 4.6 5.1 4.8 5.0 5.0 5.2 5.2 4.7 4.8 5.4 5.2 5.5 4.9 5.0
##  [37] 5.5 4.9 4.4 5.1 5.0 4.5 4.4 5.0 5.1 4.8 5.1 4.6 5.3 5.0 7.0 6.4 6.9 5.5
##  [55] 6.5 5.7 6.3 4.9 6.6 5.2 5.0 5.9 6.0 6.1 5.6 6.7 5.6 5.8 6.2 5.6 5.9 6.1
##  [73] 6.3 6.1 6.4 6.6 6.8 6.7 6.0 5.7 5.5 5.5 5.8 6.0 5.4 6.0 6.7 6.3 5.6 5.5
##  [91] 5.5 6.1 5.8 5.0 5.6 5.7 5.7 6.2 5.1 5.7 6.3 5.8 7.1 6.3 6.5 7.6 4.9 7.3
## [109] 6.7 7.2 6.5 6.4 6.8 5.7 5.8 6.4 6.5 7.7 7.7 6.0 6.9 5.6 7.7 6.3 6.7 7.2
## [127] 6.2 6.1 6.4 7.2 7.4 7.9 6.4 6.3 6.1 7.7 6.3 6.4 6.0 6.9 6.7 6.9 5.8 6.8
## [145] 6.7 6.7 6.3 6.5 6.2 5.9
```

```
## To see the first row of iris, we could use
iris[1,]
```

```
##   Sepal.Length Sepal.Width Petal.Length Petal.Width Species
## 1          5.1         3.5          1.4         0.2  setosa
```

Most of the time, the `$` is easier to work with because you can type out the name of the column and Rstudio often gives you fill in options!

```
iris$Species
```

```
##   [1] setosa     setosa     setosa     setosa     setosa     setosa
##   [7] setosa     setosa     setosa     setosa     setosa     setosa
##  [13] setosa     setosa     setosa     setosa     setosa     setosa
##  [19] setosa     setosa     setosa     setosa     setosa     setosa
##  [25] setosa     setosa     setosa     setosa     setosa     setosa
##  [31] setosa     setosa     setosa     setosa     setosa     setosa
##  [37] setosa     setosa     setosa     setosa     setosa     setosa
##  [43] setosa     setosa     setosa     setosa     setosa     setosa
##  [49] setosa     setosa     versicolor versicolor versicolor versicolor
##  [55] versicolor versicolor versicolor versicolor versicolor versicolor
##  [61] versicolor versicolor versicolor versicolor versicolor versicolor
##  [67] versicolor versicolor versicolor versicolor versicolor versicolor
##  [73] versicolor versicolor versicolor versicolor versicolor versicolor
##  [79] versicolor versicolor versicolor versicolor versicolor versicolor
##  [85] versicolor versicolor versicolor versicolor versicolor versicolor
##  [91] versicolor versicolor versicolor versicolor versicolor versicolor
##  [97] versicolor versicolor versicolor versicolor virginica  virginica
## [103] virginica  virginica  virginica  virginica  virginica  virginica
## [109] virginica  virginica  virginica  virginica  virginica  virginica
## [115] virginica  virginica  virginica  virginica  virginica  virginica
## [121] virginica  virginica  virginica  virginica  virginica  virginica
## [127] virginica  virginica  virginica  virginica  virginica  virginica
## [133] virginica  virginica  virginica  virginica  virginica  virginica
```

```
## [139] virginica  virginica  virginica  virginica  virginica  virginica
## [145] virginica  virginica  virginica  virginica  virginica  virginica
## Levels: setosa versicolor virginica
```

R will do operations along entire columns. To multiply all `Sepal.Length` values by 2, for example:

```
iris$Sepal.Length * 2
```

```
##   [1] 10.2  9.8  9.4  9.2 10.0 10.8  9.2 10.0  8.8  9.8 10.8  9.6  9.6  8.6 11.6
##  [16] 11.4 10.8 10.2 11.4 10.2 10.8 10.2  9.2 10.2  9.6 10.0 10.0 10.4 10.4  9.4
##  [31]  9.6 10.8 10.4 11.0  9.8 10.0 11.0  9.8  8.8 10.2 10.0  9.0  8.8 10.0 10.2
##  [46]  9.6 10.2  9.2 10.6 10.0 14.0 12.8 13.8 11.0 13.0 11.4 12.6  9.8 13.2 10.4
##  [61] 10.0 11.8 12.0 12.2 11.2 13.4 11.2 11.6 12.4 11.2 11.8 12.2 12.6 12.2 12.8
##  [76] 13.2 13.6 13.4 12.0 11.4 11.0 11.0 11.6 12.0 10.8 12.0 13.4 12.6 11.2 11.0
##  [91] 11.0 12.2 11.6 10.0 11.2 11.4 11.4 12.4 10.2 11.4 12.6 11.6 14.2 12.6 13.0
## [106] 15.2  9.8 14.6 13.4 14.4 13.0 12.8 13.6 11.4 11.6 12.8 13.0 15.4 15.4 12.0
## [121] 13.8 11.2 15.4 12.6 13.4 14.4 12.4 12.2 12.8 14.4 14.8 15.8 12.8 12.6 12.2
## [136] 15.4 12.6 12.8 12.0 13.8 13.4 13.8 11.6 13.6 13.4 13.4 12.6 13.0 12.4 11.8
```

Exercise (3 minutes):

- print all values of `Petal.Width` to the terminal
- You can ask for help in R using the `?` function. Try running `?mean` to find out what the `mean()` function does.
- what is the mean value of `Petal.Width`?
- what is the sum of all `Petal.Width`? (HINT: guess the function name, or google for it!)
- what is the standard deviation of `Sepal.Length`?

Packages in R

In short, R packages are usually a collection of R functions, complied code and sample data. A package bundles together code, data, documentation, and tests, and is easy to share with others. Often, the chances are that someone has already solved a problem that you're working on, and you can benefit from their work by downloading their package.

In order to be able to use a package, you first need to install (aka download) it using `install.packages()`. You only need to do this once for any package you install. To actually use functions you will have to load the package using `library()` function. You have to load libraries once at the beginning of any time you use an R script.

```
# This installs multiple packages that help manipulate & plot data:
#install.packages("tidyverse")

# This installs two packages that contain example datasets we will use later in the course:
#install.packages(c("gapminder", "titanic"))
```

Plotting

The package `ggplot2` is used for making plots from data.frames. The `ggplot2` package is a part of the `tidyverse` package, so you do not need to install it separately! Load the package by calling `library(ggplot2)`. Remember, you have to load libraries once per script.

```
library(ggplot2)
```

The function `ggplot()` is used to build plots from data. By calling `ggplot` alone we build a totally empty plot.

```
ggplot()
```


We tell ggplot to use a dataset by giving it a `data.frame` to the `data` argument.

```
ggplot(data = iris)
```

Lets make a scatterplot of Sepal width vs Petal width.

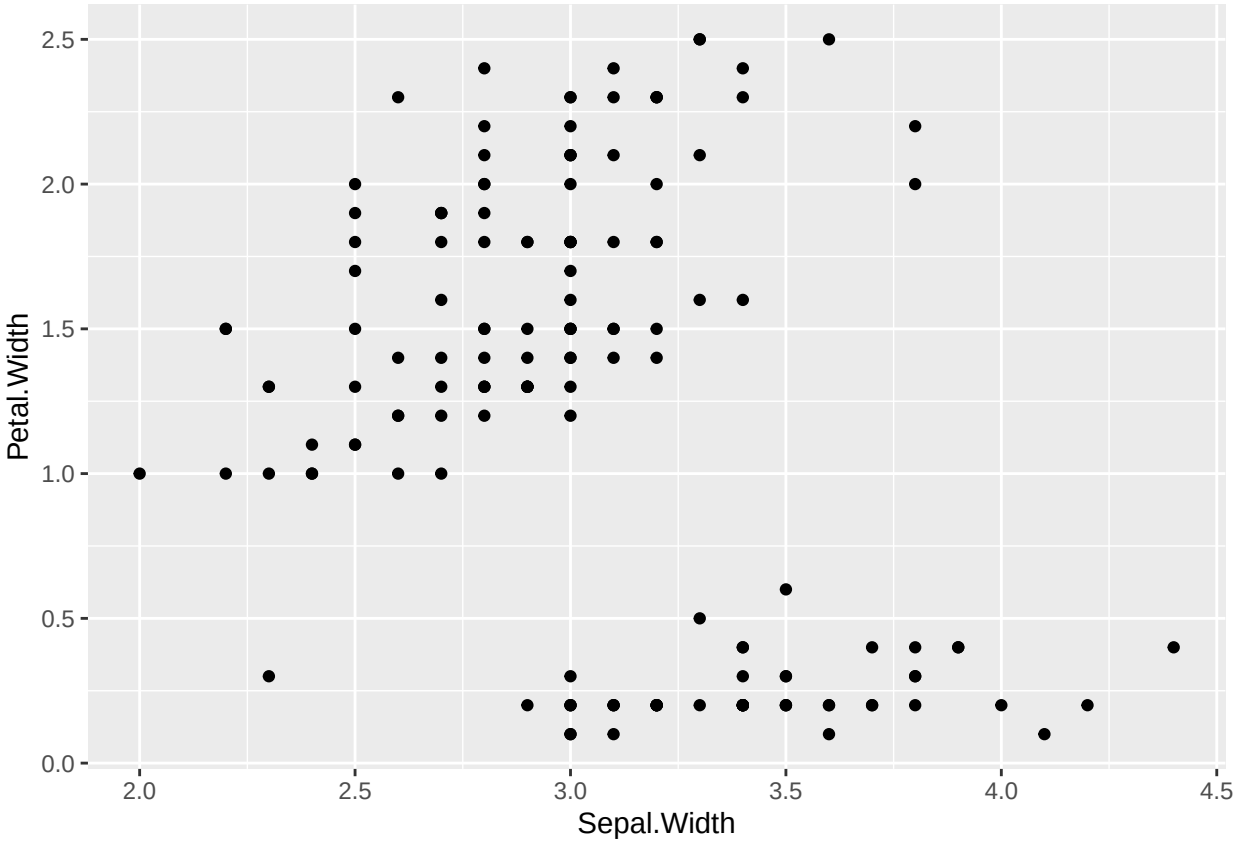
To do this we have to **map** the data onto the plot. To do this we make an **aesthetic** (with `aes()`). **aesthetics** are parts of the plot that are **determined by the data**. Here we say that we want to plot the `Sepal.Width` on the x-axis, and `Petal.Width` on the y-axis.

This creates a plotting area that is informed by the data in `iris`.

```
ggplot(data = iris, mapping = aes(x = Sepal.Width, y = Petal.Width))
```

Now let's add the points. We do this by adding a `geom`. In this case `geom_point` for datapoints.

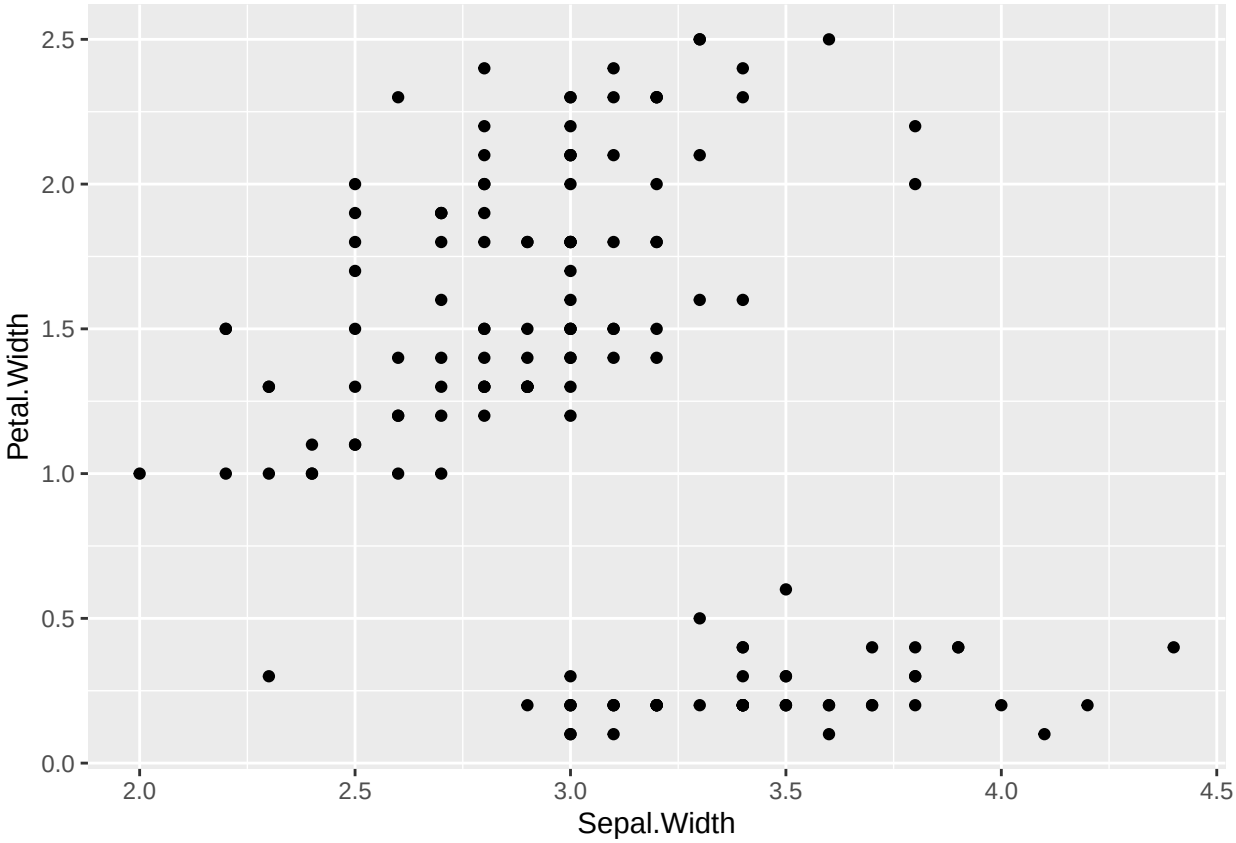
```
ggplot(data = iris, mapping = aes(x = Sepal.Width, y = Petal.Width)) +
  geom_point()
```



Plots can be saved to variables just like before

```
sepal.vs.petal <- ggplot(data = iris, mapping = aes(x = Sepal.Width, y = Petal.Width)) +
  geom_point()
```

```
sepal.vs.petal
```



We can edit the parameters of geoms. For example, you can change the color of the points by editing the `color` parameter.

```
ggplot(data = iris, mapping = aes(x = Sepal.Width, y = Petal.Width)) +
  geom_point(color = "red")
```

A really helpful cheatsheet detailing different aspects of ggplot can be found here

Exercise:

Question 1:

- Try making the same scatterplot but color the points by their `Species`.
- **Hints:**
- Try looking at the help page for `aes()` with `?aes`
- If this isn't helping, try googling a solution!

Question 2:

- Make a boxplot like the one below plotting the `Petal.Width` for each `Species`
- **Hints:**
- which `geom` adds a boxplot?
- How do you fill-in the boxplot by species?
- Try googling a solution.

More reading

R for Data Science: ggplot2 introduction - Great textbook!

ggplot2 reference manual - really useful manual with pictures!