Lecture 1

Marc Kaufmann

9/10/2019

This introduction is essentially Chapter 2 from Kieran Healy's book Data Visualization, which you should read as part of Assignment 2 and to find your way around RStudio. The reason I start with this (and continue for one or two more weeks) is that visualization is one of the more fun parts of data analysis, as well as one you should go through – so it makes for a great starting point.

We will use exclusively R Markdown files. In this case, you should simply open the file 'lecture1-to-fill.Rmd' from the repository and you are set. Whenever you create a new file, you should choose to make an R Markdown file.

Things to Know about R

There are 4 things to bear in mind about R:

1. Everything has a name
2. Everything is an object
3. You do things using functions
4. Functions come in packages

Everything has a name

Everything that you use in R has a name: variables (including datasets), functions, or special reserved words. That is the way you talk about them. Here are some examples:

```
# Numbers are called by their number, arithmetic operations by their usual symbol
2
4 + 7
7/3
7 %% 3
8 %% 3
x^2
2^2
2*2
2^3
# There are some pre-defined variables
pi
"pi"
'pi'
"pi" == 'pi'
TRUE
True
FALSE
# There are also pre-defined functions
c
"c"
'c'
"C"
```

Some Notes:

- '#' is the ____________ (EXERCISE) sign. It tells R that everything that follows is a comment and R should ignore it. Comments are there to help explain parts of the code that need additional documentation. We will overuse them initially, but reduce this as the course proceeds.
- c, "c", and "C" are not the same things
- Naming conventions: When you name variables and functions, you should use snake case.

Exercise: For each of the lines in the following code chunk, write in a comment next to it what it returns. I completed the first example for you.

```
3          # -> 3
7 %% 3     # -> 1
False      # -> Error
'pi'       # -> "pi" as a string
FALSE      # -> FALSE as boolean value
```

Exercise: Look up what 'snake case' is and add your answer here.

Answer: According to Wikipedia, snake case (or snake_case) is the process of writing compound words so that the words are separated with an underscore symbol (_) instead of a space.

Exercise: What types of names are allowed in R? Look it up and write your answer here as you understand it. Then provide 3 examples of things that are not valid names in R for different reasons.

Answer: There are special rules for naming a variable in R. I usually follow the snake_case convetion of naming variables and exluding numbers and special characters. I listed 3 mistakes you can possibly make upon naming a variable.

- Cannot start with a number:

```
1_fake_name <- "This will give an error"
```

- Cannot contain a special char such as "÷" or " " (and many others):

```
kovacs÷kornel <- "This will also give an error"
kovacs kornel <- "Same here"
```

- Cannot start with a dot(.) and a number:

```
.1kovacs_kornel <- "I am fed up with errors"
```

Everything is an object

There are built-in objects and objects you import or create. Most importantly, you assign values to objects you create with the '<-' operator: a '<' ('less than') followed by a '-' ('minus' or 'dash').

```
marcs_new_object <- "I am a fancy object"
marcs_new_object
```

One important function to create objects is the $c()$ function, which combines several items into one objects:

```
marc_new_combined_object <- c(marcs_new_object, "Some apples, because they are healthy")
marc_new_combined_object
```

In principle, you can use '=' to assign values, but this is very non-idiomatic R. I will subtract points for using '='. Use '<-'.

Class Exercise: Let's start doing something with data. How many of the following programming languages have you *heard* of?

```
# Indent the list
list_of_programming_languages <- c(
"R",
"SQL",
"Racket",
"Lisp",
"JavaScript",
"ECMAScript",
"bash",
"C",
"Perl",
"Logo",
"Scratch"
)

languages_heard_of <- c(11, 4, 5, 3, 4, 5, 5, 7, 5, 5, 5, 5, 5, 4, 4, 6, 7, 5, 6, 5, 5, 4, 8, 7, 7, 9, 5
languages_heard_of
```

```
##  [1] 11  4  5  3  4  5  5  7  5  5  5  5  5  4  4  6  7  5  6  5  5  4  8
## [24]  7  7  9  5  5  5  5  4  6
```

You do things using functions

> A function is a thing that does things to things.
>
> Me, paraphrasing Cosma Shalizi

A function is a special object that you can call, such as the function *mean*. It is an object:

```
mean
```

```
## function (x, ...)
## UseMethod("mean")
## <bytecode: 0x000000000931e358>
## <environment: namespace:base>
```

When we *call* it, we put parentheses at the end, and we tell it what to perform its action on:

```
mean(c(0,10))
```

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

```
mean(languages_heard_of)
```

```
## [1] 5.5
```

```
mean("a")
```

```
## Warning in mean.default("a"): argument is not numeric or logical: returning
## NA
```

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

Exercise: What happens if you call the function *mean* without any arguments, i.e. *mean*()?

Short answer: We get an error.

Detailed answer: R sees that the mean function has not been given any arguments. Accordingly, R tries to execute the code with default params of the function. Here comes the problem, because it has no default params so we receive an error message.

Functions come in packages

As you may start to realize, lots of functions already exist, such as *mean*() and *sd*() and *c*(). There are many more functions that were written by other people and bundled into packages so that you can use them. Many packages come bundled with R from the start (base R), while you can install others on your system via 'install.packages()' and use via 'library()'. We will soon use the 'ggplot()' function. Try using it now:

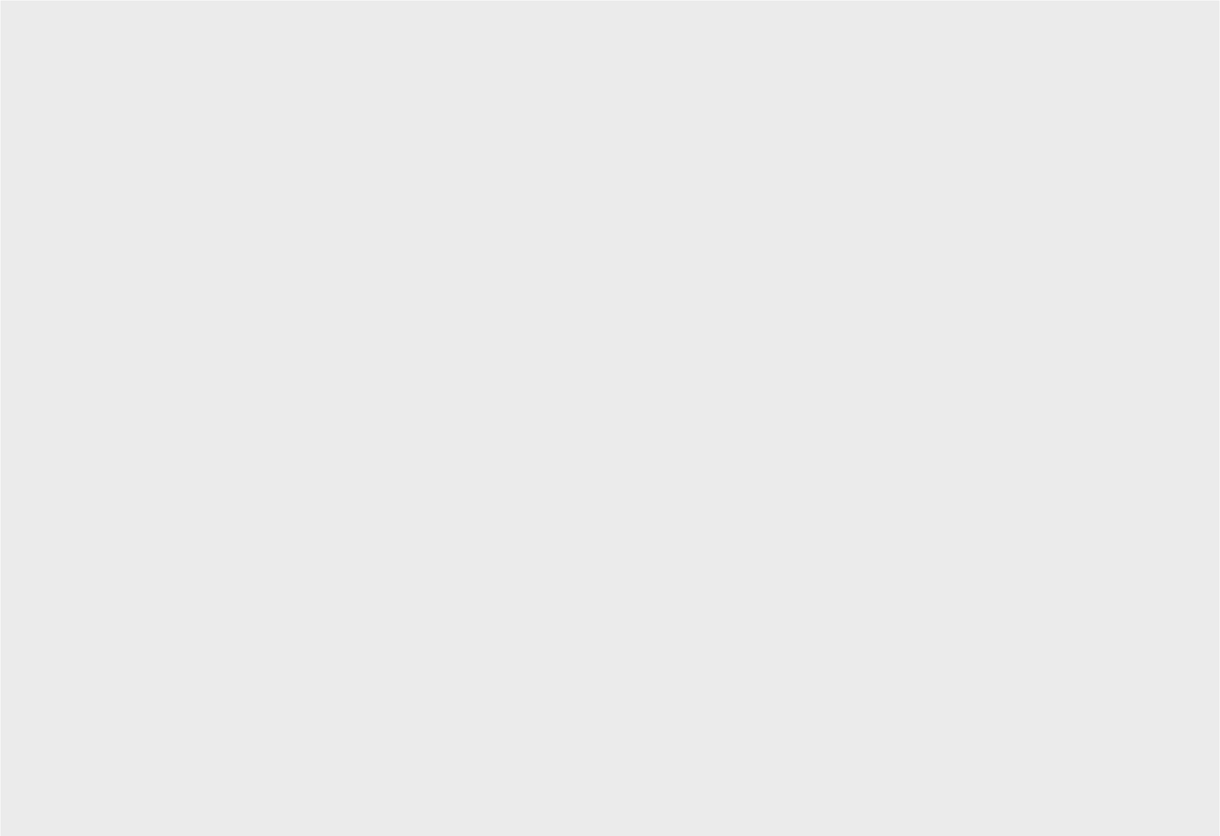
```
ggplot
```

You will get an error. We first need to import the library:

```
library(ggplot2)
ggplot
```

```
## function (data = NULL, mapping = aes(), ..., environment = parent.frame())
## {
##     UseMethod("ggplot")
## }
## <bytecode: 0x000000000797a328>
## <environment: namespace:ggplot2>
```

```
ggplot()
```



This tells R to go and get all the objects defined in the library called ggplot2, which includes *ggplot*. This gives us our first data (non-)visualization: a beautifully empty plot.

Think of it as the equivalent of recipe books: you can either experiment and concoct your own recipes, which is fine if you are a cook. Alternatively, you get recipe books from great chefs and follow their instructions. Depending on the type of recipe, you still need a lot of background knowledge – what does it mean to boil, fry, blanch, roast – but you don't have to figure out everything.

How to figure out what is what

Suppose you have an object x that you don't know what it is. You can do a few things to find out:

```
x <- c(1, 3.0, 2.9)
x
```

```
## [1] 1.0 3.0 2.9
```

```
str(x)
```

```
##  num [1:3] 1 3 2.9
```

```
y <- c("1", "3.0", "2.9")
str(y)
```

```
##  chr [1:3] "1" "3.0" "2.9"
```

```
y
```

```
## [1] "1"   "3.0" "2.9"
```

```
class(x)
```

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

```
z <- c("1", 1)
z
```

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

```
c("thing", 1010)
```

```
## [1] "thing" "1010"
```

```
list("thing", 1010)
```

```
## [[1]]
## [1] "thing"
##
## [[2]]
## [1] 1010
```

```
c("thing", 10, function(x) {x})
```

```
## [[1]]
## [1] "thing"
##
## [[2]]
## [1] 10
##
## [[3]]
## function(x) {x}
```

For built-in functions, you can also ask for help or bring up the documentation with 'help()' or '?' or '??':

```
help(summary)
```

```
## starting httpd help server ... done
```

```
summary(languages_heard_of)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##     3.0     5.0     5.0     5.5     6.0    11.0
```

```
?summary
```

Exercise: What do ‘class()’ and ‘str()’ do? Use ‘help’ (or ‘?’). Don’t spend too much time reading the docs. Which description do you find more helpful?

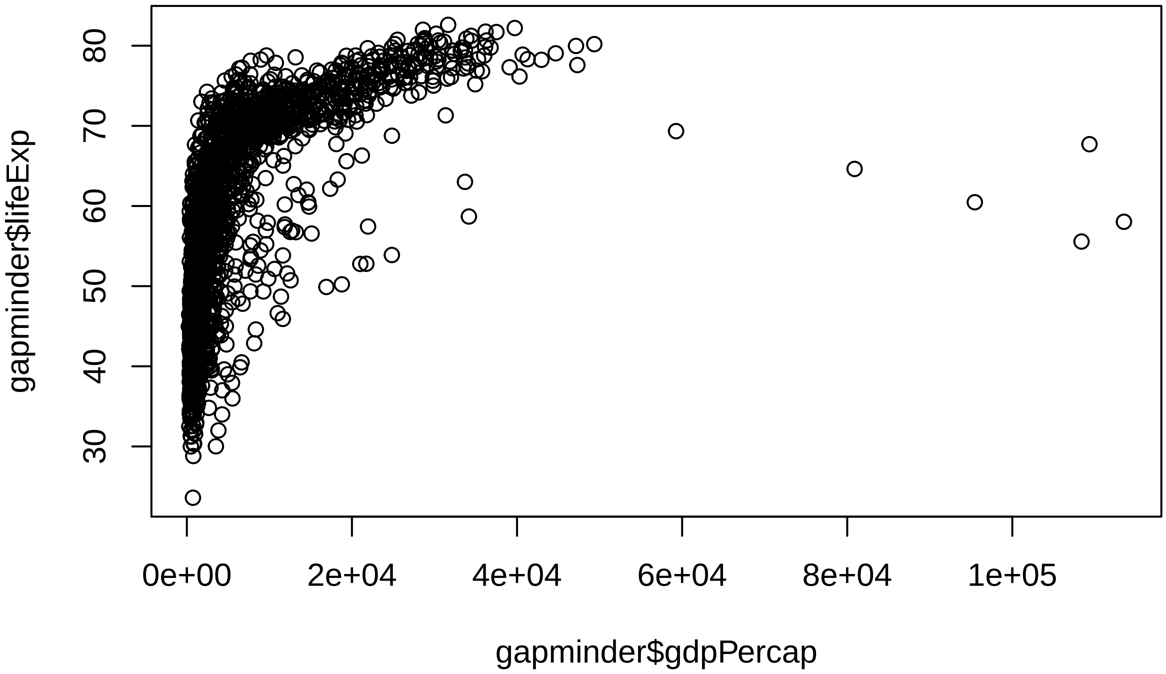
Answer: In a way, they both return the structure of an object. As I see, ‘str()’ gives you more details than ‘class()’. E.g. when calling them with an argument such as ‘c(5,6,7,8)’, then ‘class’ only concentrates on the inner values of the function ‘c’ and does not care about the function itself. I prefer to have more details, accordingly I will use ‘str()’.

Make Your First Figure

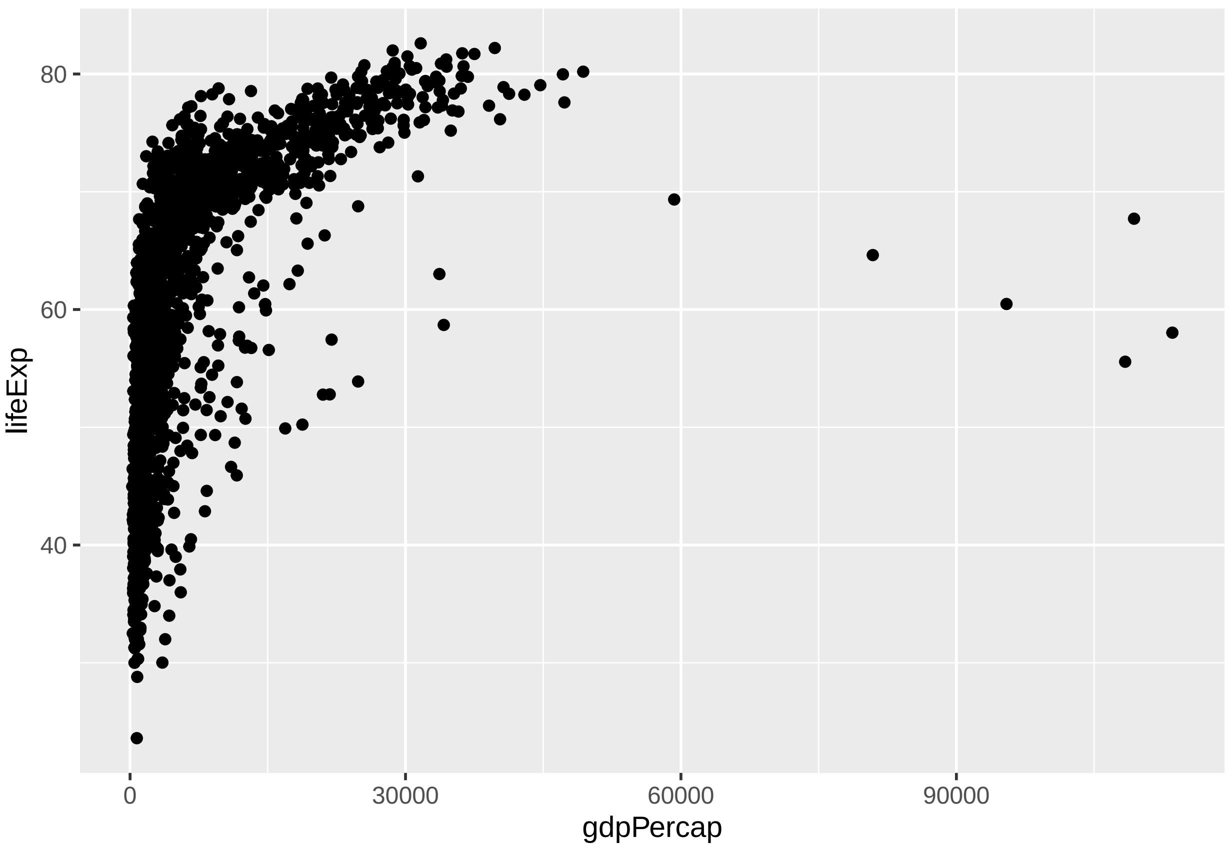
```
library(gapminder)
gapminder
```

```
## # A tibble: 1,704 x 6
##    country     continent  year lifeExp      pop gdpPercap
##    <fct>       <fct>     <int>   <dbl>    <int>     <dbl>
##  1 Afghanistan Asia       1952    28.8  8425333      779.
##  2 Afghanistan Asia       1957    30.3  9240934      821.
##  3 Afghanistan Asia       1962    32.0 10267083      853.
##  4 Afghanistan Asia       1967    34.0 11537966      836.
##  5 Afghanistan Asia       1972    36.1 13079460      740.
##  6 Afghanistan Asia       1977    38.4 14880372      786.
##  7 Afghanistan Asia       1982    39.9 12881816      978.
##  8 Afghanistan Asia       1987    40.8 13867957      852.
##  9 Afghanistan Asia       1992    41.7 16317921      649.
## 10 Afghanistan Asia       1997    41.8 22227415      635.
## # ... with 1,694 more rows
```

```
plot(gapminder$gdpPercap, gapminder$lifeExp)
```



```
p <- ggplot(data = gapminder,
            mapping = aes(x = gdpPercap, y = lifeExp))
p + geom_point()
```



As you can see, some datasets are conveniently bundled for you as libraries.

Assignment 2

This assignment is due before the start of class 2. Commit your knitted assignment2.html to your git and push it to your repository regularly as you make progress.

- Only your work until _____________ (EXERCISE) will be considered.
- If you know that you might miss the deadline, you have to email me in advance and I will tell you whether you can get an extension. No extension will be granted less than 24 hours prior to the deadline – you should have started working on it.
- If you miss the deadline (even by 1 minute) you lose 25% irrespective of technical issues such as 'the internet went down'. It wasn't down the whole week, and you knew a deadline was coming. In a business setting, you would probably lose more than 25%. Repeat offenders will face an increasingly large penalty.
- If you struggle *answering* the assignment, you should submit what you tried, and send me a message that you struggled. Much of the grade initially is on trying out things, even if it doesn't work out, so you should submit.

Part 1 - DONE

Read chapter 2 of Data Visualization. Make sure you know where the console is, the editor, and what code chunks are in RStudio.

Part 2 - DONE

Complete all the exercises in the lectures notes of lecture 1. Put your answer to an exercise right after the question of the exrcise in the R Markdown file. When done, knit the document, commit the changes with the commit message "Part 1 of assignment 2" and push them to your GitHub repository.

Part 3 - DONE

Fill in the holes in the lecture notes of lecture 1, if you didn't complete it during class. When done, knit the document, commit the changes with the commit message "Part 3 of assignment 2" and push them to your GitHub repository.

Knit the slides for class 1 to html and/or pdf, commit the .Rmd and .html files – but not the pdf – and push. If the pdf is hard to get working (due to weird error messages), see whether you can find out *what* the problem is (Stackoverflow, Discourse. . .) without trying to solve it yet. Pdf and Latex issues can be. . . interesting shall we say.

Note: If you can't figure out how to use git, post a question on the discourse forum and send me the html file by email.

Part 4 - DONE

Track as many error messages as possible that you made during this week (stop once you get to 4). Put the code that caused the error in the chunk below, and copy the error message as a comment below it. I provide an example.

```
# My first example is on Discourse. Concerning the nature of the problem, I am not able to insert that

# "7'
# Error: unexpected numeric constant in: "7'

# "k" + "o" + "k" + "o"
# Error in "k" + "o" : non-numeric argument to binary operator.
# In Python, this works. It results in a new string: "koko", but in R, I just get an error.

# This is not a syntax error, but still an error. My brain is just used to pushing CTRL+S all the time.
```

Post one of these 4 errors on discourse.

Part 5 - DONE

Try fo figure out two of the following and post your answers on Discourse for others to read:

- Identify elements from my *eureka_or_bust* example in the slides and see whether you can figure out what the different elements mean to R.
- What does *eval=FALSE* mean in part 4? Figure this out by replacing it by *eval=TRUE* and seeing what you get.
- What is the Tidyverse?
- Who is Hadley Wickham?

Resources for this lecture