DATA 100,Week 1 (A)

Set up

The Textbook(s)

The textbook(s) we will follow is a combination of the two versions of the same book, both of which are freely available online.

1. *R* for Data Science (1e), 2017.
2. *R* for Data Science (2e), 2023 .

The language *R* and the packages in tidyverse evolved quite a bit between the two versions. We will mostly follow the second version for this reason.

We will go through the material in slightly different order from how the textbook does it.

DATA 100,Week 1 (A)

- For the technicalities (language, packages, functions), which is more up-to-date.

We will follow another book on modelling basics:

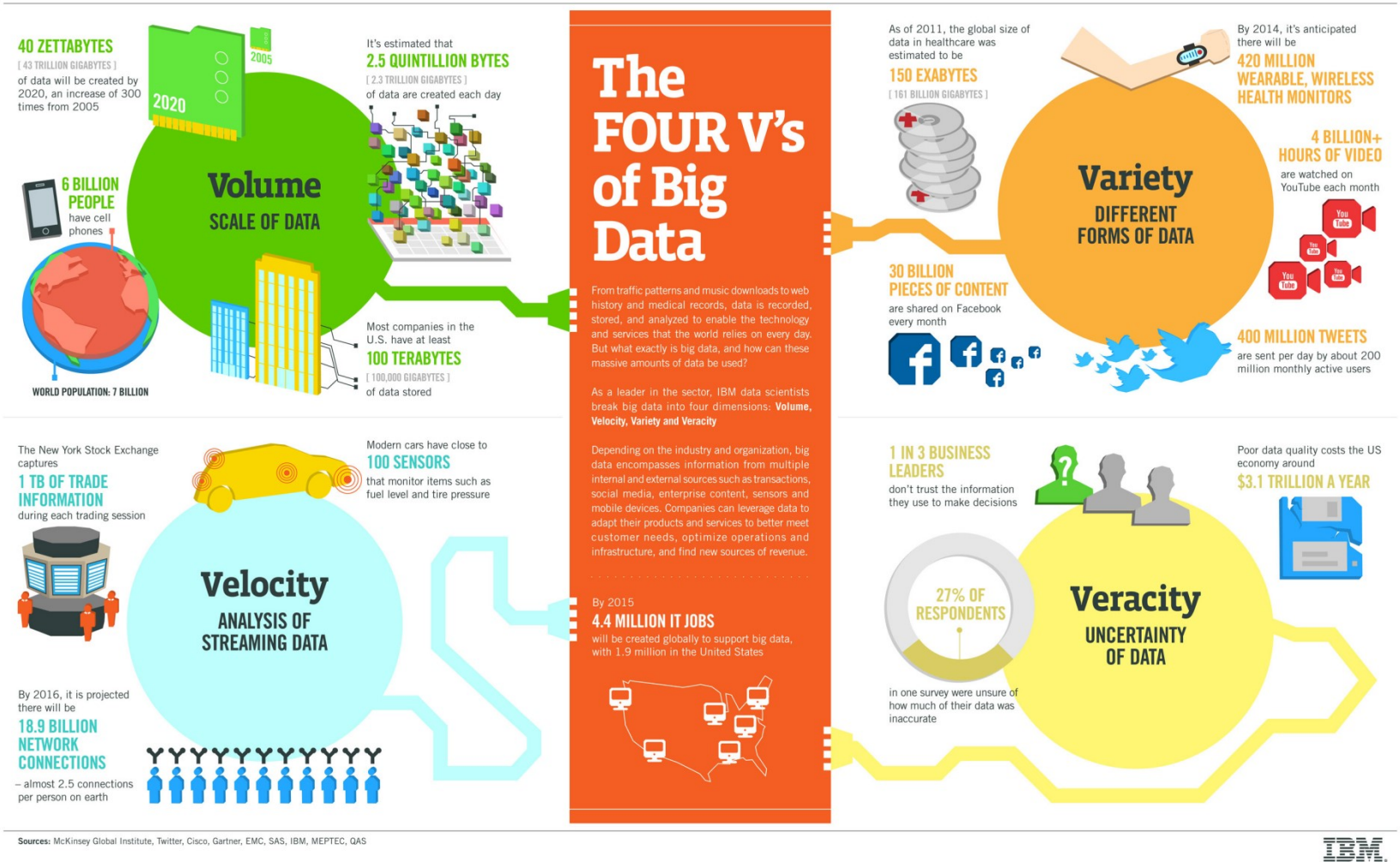
- Tidy modelling with *R*.

In summary, we will try to cover the majority of the second edition (may not in the same order), together with the modelling basics part of Tidy modelling with *R*.

- and supplement it with discussions of descriptive statistics, to understand the output better.

DATA 100,Week 1 (A)

What is in Data Analytics



DATA 100,Week 1 (A)

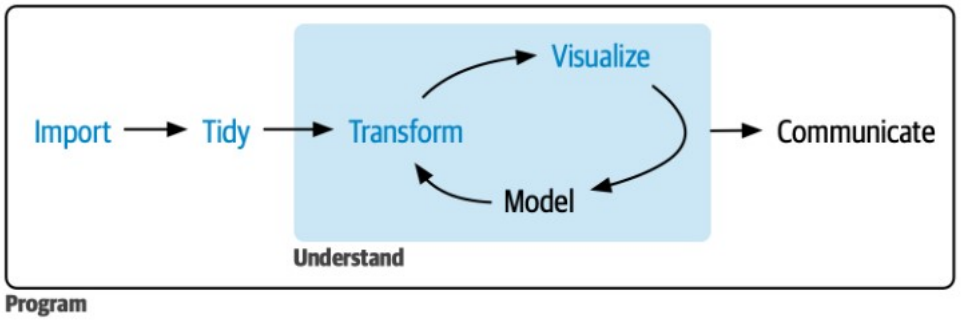
- ***Volume and Velocity*** demand computational power and knowledge
- ***Variety*** creates needs for different methods, so that one-size cannot fit all
- ***Veracity*** is very tricky, as being played out everyday as we speak

DATA 100,Week 1 (A)

Data v.s. oil

Operation stages	**Oil**	**Data**
Prospecting	Look for it in the ground, physical work	turn everything into data, physical work
Extraction	Take it out from the ground	measure and record, networks, apps, accounts
Preparation	be ready for refinery	be ready for analysis, make tidy, clean ...
Refinement	into usable oil products	into usable data products, figures, diagrams, reports ...
Downstream	pesticides, plastics, cosmetics, pollution ...	software, newspaper, website, scientists, decision makers ...
Consumer	more stuff (more waste)	public opinions, regulations, organizations ...

DATA 100,Week 1 (A)



Data project work flow

Data Analysis (for this course at least) mostly concentrates on Preparation and Refinement

DATA 100,Week 1 (A)

Data science

- Look at data from the point of view of a scientist:
  - ✓ do experiments,
  - ✓ make hypothesis,
  - ✓ test hypothesis by more experiments,
  - ✓ rinse and repeat,
  - ✓ arrive at some conclusion

DATA 100,Week 1 (A)

- Look at decision making of all kinds from the point of view of data:

  ✓ tell / refute a story with data,
  ✓ make / break a decision by data,
  ✓ persuade / disillusion using data, ...

Data science is a tool.

The correctness of the conclusion not only depends on the correctness of the inputs, but also depends on interpretation.

It is also important that the tool is used responsibly.

DATA 100,Week 1 (A)

- ***The science:*** math, statistics, computer programming, etc.
- ***The output:*** figures, diagrams, reports, presentation, etc.
- ***The hope:*** try to be useful, and hopefully helpful.

DATA 100,Week 1 (A)

To get you excited

From Wikipedia:

- Hans Rosling (27 July 1948 - 7 February 2017) was a Swedish physician, academic, and public speaker held presentations around the world, in which he promoted the use of data to explore development issues.

DATA 100,Week 1 (A)

Data need statistics

- Many freely available datasets in the wild

  ✓ Kaggle
  ✓ Open Government
  ✓ CIA
  ✓ World Bank
  ✓ COVID-19
  ✓ Weather / climate related
  ✓ …

DATA 100,Week 1 (A)

- Many more datasets in private companies

  - ✓ Alphabet (parent of Google)
  - ✓ Meta (previously Facebook)
  - ✓ Musk's X (previously Twitter)
  - ✓ Amazon (?)
  - ✓ ISPs
  - ✓ Cellular providers
  - ✓ Banks

DATA 100,Week 1 (A)

Data need programming

From Wikipedia:

R is a programming language for statistical computing and graphics supported by the *R* Core Team and the *R* Foundation,

- tidyverse and tidymode1s are suites of extension packages that this course will be mostly work with

- together with a few other packages that will be included as we go

In the following, we'll start with the most elementary part of the *R* language

DATA 100,Week 1 (A)

Arithmetic

The arithmetic in R is straightforward

```
5 + 3 / 4# mixed arithmetic operation -- recognizes
order of operations
#> [1] 5.75
4^3 # cube of 4
#> [1] 64
```

DATA 100,Week 1 (A)

```
2^(1/2) # square root of 2
#> [1] 1.414214
log(3) # NATURAL log (base e) of 3
#> [1] 1.098612
log10(5) # log (base 10) of 5
#> [1] 0.69897
pi # in language constant, 3.14159....
#> [1] 3.141593
cos(pi) # trig function, angles in radian -- pi = 180
degrees
#> [1] -1
sin(pi) # issue with limited precision
#> [1] 1.224647e-16
```

DATA 100,Week 1 (A)

Notice: log, log 10, cos and sin has () following them containing another value. They are examples of a function, which will be briefly described below.

There are different types of objects in R. For instance

- 2 L is an integer (the L specifies that 2 is an integer)
- 3.4 is a double (real number, writing 2 without L following it makes it a double as well)
- ' a ' is a character (' abd' is a string)
- TRUE is a boolean value (as is FALSE)

DATA 100,Week 1 (A)

The following expressions involve ==, an boolean operator

- which produces TRUE if both sides are the same
- and FALSE if they are not the same
- ... there are always more details, but we can wait till they actually show up and ! =, which is the opposite to ==.

```
2 + 2 == 5
#> [1] FALSE
2 + 2 != 5
#> [1] TRUE
```

DATA 100,Week 1 (A)

Variables are useful in expressing relations and describing procedures. In the simplest situation, they help with repetitive work.

```
#assign values to variables, or `names`

x = 3

#= sign can be used

y <- -1 #we will prefer <- in this course
```

If the following needs to be computed for more than one collections of x and y values, changes need only to happen in the assignment block above.

DATA 100,Week 1 (A)

```
x + y
#> [1] 2
x * y
#> [1] -3
x^y - y^x
#> [1] 1.333333
3*x^2 - 2*x*y + y^2
#> [1] 34
```

DATA 100,Week 1 (A)

A variable can be thought of as an abstract box, whose content can change based on circumstances

- can be assigned more complicated objects, not only numbers
- They are most useful when we start working with functions - which is really all that we do

Type the variable name in a code block and run it shows the value of the variable in the output, or generates an error message if the variable is not defined previously.

```
x
#> [1] 3
p
#> Error in eval(expr, envir, enclos): object 'p' not found
```

DATA 100,Week 1 (A)

Vectors

Similar to thinking of a variable as a box, a vector can be thought of as a collection of boxes

- which hold similar type of things
- A vector is constructed with c() as follows

```
c(-1,3,4.4)
#a vector of numbers (double)
#> [1] -1.0 3.0 4.4
c(TRUE, TRUE, FALSE, FALSE) #a vector of boolean values
#> [1] TRUE TRUE FALSE FALSE
c('a', 'c', 'd')
#a vector of character values
#> [1] "a" "c" "d"
```

DATA 100,Week 1 (A)

A variable can be a vector. The typeof a vector variable is the type of the entries in it.

- There are some subtleties in determining the type, which we will come back to later

```
num <- c("-1",'3','4.4')
typeof(num)
#> [1] "character"
'4.4' + '3'
#> Error in "4.4" + "3": non-numeric argument to binary operator
```

DATA 100,Week 1 (A)

A similar but more versatile collection object is a list,which can contain absolutely anything, even a list of functions:

```
trig <- list(sin, cos, tan) #it has the same effect as c(sin, cos, tan)
typeof(trig)
#> [1] "list"
typeof(sin)
#> [1] "builtin"
```

DATA 100,Week 1 (A)

Functions

The functions are the main workhorse of a programming language.

- We have seen the mathematical functions such as sin and log
- and typeof is also a function in R, that describes the type of an object

The basic idea of a function is quite straightforward

- It takes inputs - called parameters, or arguments
- The internals of a function precisely describe a process of manipulating these inputs
- ... which can be treated as a blackbox, as long as we understand what it does in plain English
- It produces some outputs, or use the inputs to perform some other tasks
- "programming changes the world one function at a time"

DATA 100,Week 1 (A)

We have already seen how to call a function, and here are some further examples.

- ✓ sqrt is the name of the square root function
- ✓ it takes one numeric parameter and generates one numeric output
- ✓ It is called using its name
- ✓ followed by the parameter in round parenthesis ()

```
sqrt(4)
#> [1] 2
```

DATA 100,Week 1 (A)

The output of a function can be assigned to another variable

```
x = sqrt(4)
#or
y <- sqrt(3)
```

The following code does the assignment and displaying of result at the same time, by putting the assignment code into a pair of round parenthesis ()

```
(x <- sqrt(4))
#> [1] 2
```

DATA 100,Week 1 (A)

sum is a function that takes many parameters, and compute the sum of all of them

```
sum(3,2,-3,7)
#> [1] 9
```

DATA 100,Week 1 (A)

When there are more than one parameters to a function

- they are separated by, in () that follows the name of the function
- the order of including the parameters in () matters (pass by position)
- for R, if allowed, one can also pass the parameter by name, we will see such examples very soon.

We will see further examples of calling (invoking) functions in R below.
What if something is wrong when calling a function?

- there will generally be an error message
- which helps us understand the situation and correct the error

Debug: the process of making and correcting mistakes

```
sum(3, 4)
#> [1] 7
```

DATA 100,Week 1 (A)

Two short R functions

We will give two examples of simple R functions. More details on how to make our own functions will be discussed later in the course.

- These are to give you a taste of what you would be able to accomplish with simple functions
- For the most part (at least in the beginning), we will call existing functions, instead of writing our own

Question: Find the sum of the squares of the first k positive integers
This problem has a known answer in the form of a mathematical formula, which can be put into a function as follows:

DATA 100,Week 1 (A)

```
formula_square_sum <- function(k) { #k is the name of the parameter
k*(k+1)*(2*k+1) / 6 #directly write down the formula and output the result
}
```

Test:

```
formula_square_sum(3) #1 + 4 + 9
#> [1] 14
```

DATA 100,Week 1 (A)

The nice thing about programming is that one may brute force it, using the R function sum()

```
R_square_sum <- function(k) {
sum((1:k)^2)
#1:k creates a vector containing the first k positive integers, i.e. (1, 2, ..., k)
#^2 raises each term in the vector to power 2 (i.e. square)
#sum then adds all the entries in the vector together
}
```

DATA 100,Week 1 (A)

Caution: Mathematically, the operation $(1,2,3,4,5)^2$ does not make sense as is, let alone writing $(1,2,3,4,5)^2 = (1,4,9,16,25)$. It would be patently wrong if one attempts this on a math assignment or test, without properly defining it first. It is a programming shorthand introduced by R, and really only works this way in R. For instance, Python does not have such a shorthand.

Or directly write a for loop (will be discussed later in the course)

DATA 100,Week 1 (A)

```
square_sum <- function(k) {
num = 1:k
#this create a vector containing the first k positive integers
s = 0
#this will contain the sum of all the squares of the integers in num
for (i in num) { #this is called a `for loop`
s = s + i^2
#which goes through each integer in the num vector and add the square to s
}
s
#now the resulting s is the result we need
}
```

DATA 100,Week 1 (A)

Test:

```
R_square_sum(6)
#> [1] 91
square_sum(6) #1 + 4 + 9 + 16 + 25 + 36
#> [1] 91
```

DATA 100,Week 1 (A)

Then we can see that all the ways give the same answer

```
k <- 6
#an arbitrary value
square_sum(k) == R_square_sum(k)
#> [1] TRUE
square_sum(k) != formula_square_sum(k) #compare if the two functions give the same answer
#> [1] FALSE
```