Data sources, databases and SQL

Day 6

Solveig Bjørkholt

4. July

Plan for today

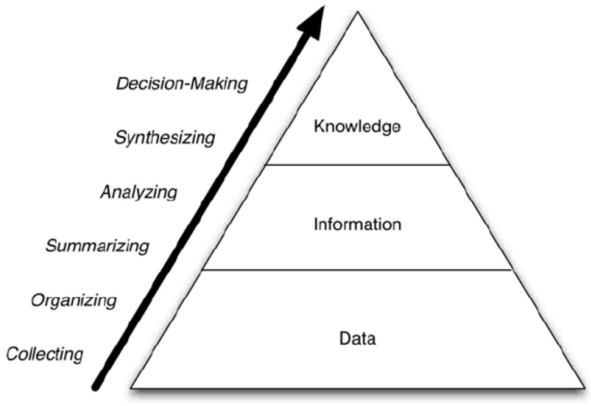
What we will learn today:

- Different types of data
- Data formats
- Databases
- Publicly available datasets

Different types of data

What is data, exactly? Data is a way to store and transfer information. It can be text, it can be numbers, it can be speech, it can be a lot of things. When data is saved on a computer, it's bits and bytes in the memory.

To humans, data is often viewed as a collection of facts. Collections of facts are important, but they are not always useful. The reason is that human beings are unable to process large amounts of facts and make sense of it without some sort of system. That is why we have methodology – strategies to make sense of data in order to produce insight and knowledge. Typically, the work process with regard to using data looks something like on the figure.

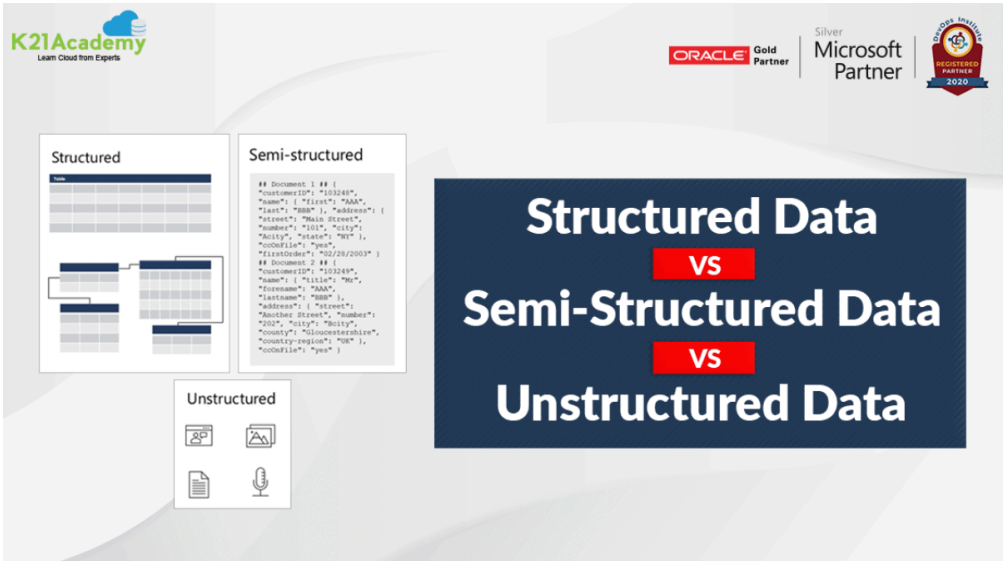


How to collect data? Well, you could for example send out a survey to respondents and ask them to reply to a set of questions. You could choose an object of observation and note down things like temperature

or number of swear words per day. You could conduct interviews or have focus groups. You could have some sort of monitor that stores info on e.g. transactions. Notice that all these methods simply take some information an record it. The recorded information – that's data.

This means that data is more than we conventionally deem as "data". Traditionally, we've been used to thinking of data as a spreadsheet with observations and variables. However, data comes in many different categories. We often distinguish between (1) structured, (2) semi-structured and (3) unstructured data.

The good thing about structured data is that is is vastly easier to analyze it. The good thing about unstructured data is that there is so much of it.



Structured data

Structured data is data with a predefined format. Often, when we think of "data", we think of structured data. It generally comes in tabular form that is represented by columns and rows. All rows in the table have the same set of columns. This includes if the data has some missing value, indicated by `NA` (not available). In the example below, the dataset contains three variables, `id`, `name` and `age`, and all the observations in the dataset – people – have information filled out for each variable.

```
## # A tibble: 3 x 3
##      id name              age
##   <dbl> <chr>           <dbl>
## 1     1 James Kirk         40
## 2     2 Jean-Luc Picard    45
## 3     3 Wesley Crusher     NA
```

When we have many structured datasets together in a database, we call them "relational data". In that case, we can have many tables for the same set of observations, and indicate that it is the same set of observations through the `id` variable.

Semi-structured data

Semi-structured data lies somewhere between structured and unstructured data. There is some structure to it, but not in a tabular way. Each observation does not necessarily have the same number of variables.

Examples of this type of data includes XML-files and JSON-files. Instead of being defined in a tabular form, they are defined in "key-value" pairs. The "key" is the variable and the "value" is the data point for that observation.

XML

```
<empinfo>
   <employees>
      <employee>
         <name>James Kirk</name>
         <age>40></age>
      </employee>
      <employee>
         <name>Jean-Luc Picard</name>
         <age>45</age>
      </employee>
      <employee>
         <name>Wesley Crusher</name>
         <age>27</age>
      </employee>
   </employees>
</empinfo>
```

JSON

```
{  "empinfo" :
     {
          "employees" :  [
          {
               "name" : "James Kirk",
               "age" : 40,
          },
          {
               "name" : "Jean-Luc Picard",
               "age" : 45,
          },
          {
               "name" : "Wesley Crusher",
               "age" : 27,
          }
                              ]
     }
}
```

Unstructured data

Unstructured data does not have a predefined model for how the computer should understand and distinguish between variables, units and observations. Often, they seem rather text-heavy, but they can also include other elements such as numbers, dates and pictures. This includes all finds of data in their native formats, including documents, audio-files, video-files, emails, sensory data, and so on.

Working with unstructured data requires a lot of pre-processing, and the task can seem rather daunting at times. You need to find a way to extract the information you need from the unstructured data and use it in a structured fashion. This can either be done through structuring, or through using methods especially designed for unstructured data, such as many techniques used in unsupervised machine learning (which we will come back to).

Data formats

We have already been through a few data formats. Basically, they tell us how information is stored. At the end of your file, there is a dot and a name. This suffix tells you the data format. For example, word documents typically end with `.doc`, and excel-files often end with `.xlsx`.

The suffix on files tell us what format they have. Some examples of file formats include:

Format	Suffix	Data type
R data-file	.RData	Structured
R data-file	.RDS	Structured
Stata data-file	.dta	Structured
Database-file	.db	Structured
CSV-file	.csv	Semi-structured
JSON-file	.json	Semi-structured
XML-file	.xml	Semi-structured
HTML-file	.html	Semi-structured

Format	Suffix	Data type
Excel-file	.xlsx	Semi-structured
Word-document	.doc	Unstructured
Pdf-document	.pdf	Unstructured
Picture	.jpeg	Unstructured
Audio-file	.mp3	Unstructured
Media-file	.mp4	Unstructured

These are examples to give an overview, but the table above is by no means the whole story. As you probably know, pictures can for example come in `.png` just as they come in `.jpeg`.

To work with these files in R, we need to use different functions. The format decides which function that works.

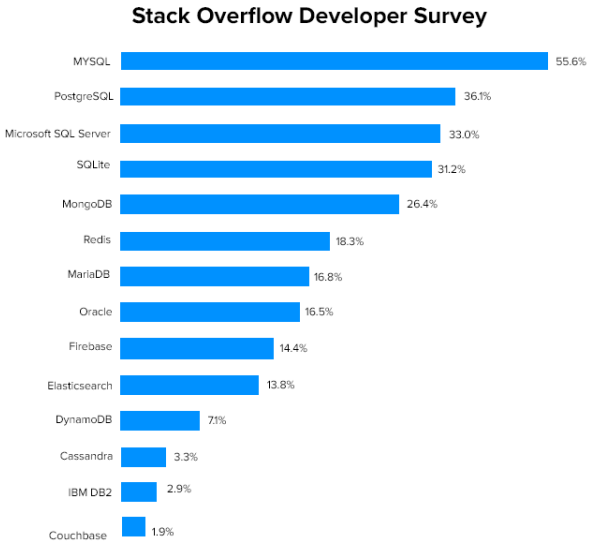
```
# R-data files
load("mydata.RData")
df <- readRDS("mydata.RDS")

# Stata file
library(haven)
df <- read_dta("mydata.RData") # Stata-file

# Database file using SQLite
library(RSQLite)
con <- dbConnect(SQLite(), "mydatabase.db")

# csv-file
df <- read_csv("mydata.csv")

# json-file
library(jsonlite)
df <- read_json("mydata.csv") %>% flatten()

# xml-file
library(XML)
df <- xmlParse("mydata.xml") %>% xmlToList()

# html-file
library(rvest)
df <- read_html("mydata.html") %>% html_node("body") %>% html_text2()

# excel-file
library(readxl)
df <- read_excel("mydata.xlsx")

# word document
library(readtext)
df <- readtext("mydata.doc")

# pdf-document
library(readtext)
df <- readtext("mydata.pdf")
```

```
# jpeg-file
library(jpeg)
df <- readJPEG("mydata.jpeg")
```

Databases

When we large amounts of structured data, we often place it in a database. In companies that rely on data (which is a large amount of companies in these days), it is very normal to have databases. One database might for example contain lots of tables detailing information on the work-life on Norwegian inhabitants, including one table on salaries and one on absence. The unit in these tables would be the same – Norwegian inhabitants – which makes the tables relational. You could merge them and create one big table with all variables. However, since this would overload both us and the computer with unnecessary information and make everything slower, we split it up into tables and pick and choose what we need from each table instead.

There are several different databases, for example MySQL, Oracle and PostgreSQL. All of these databases are relational. Then we also have non-relational databases such as MongoDB, where the relationship between different data sources is shown in a different way. Non-relational databases work better for semi-structured and unstructured data. We are not going to go into the details on these different databases here, but it's good to know that they exist.



SQLite

Databases can be large-scale and tricky, but they don't have to be. There is a smooth way into the world of databases, and that is SQLite. SQLite is a program that allows you to create databases for yourself, your friends and others who might be interested, and it's quite well integrated with R.

We will give an example on how to create your own database using SQLite. First, we create two datasets to add to the database.

```
dataset_salaries <- as_tibble(list(id = c(1, 2, 3, 4, 5),
                                   age = c(43, 53, 25, 37, 41),
                                   occupation = c("carpenter", "doctor", "accountant", "priest", "resep
```

```
                                    salary = c(5300, 6800, 2100, 1600, 1300)))

dataset_absence <- as_tibble(list(id = c(1, 2, 3, 4, 5),
                                  age = c(43, 53, 25, 37, 41),
                                  absence_this_year = c(1, 0, 0, 0, 1)))
```

Then, we need to get the two packages `DBI` to work with databases in R and `RSQLite` to work with SQLite in R. If this is the first time you use these packages, remember to install them first using `install.packages`.

```
library(DBI)
library(RSQLite)
```

In the next step, we create a database and save it on our computer. If you want to just save it in the short-term memory and do not care about whether you save your data in the long run, you can write "':memory:""in the last argument.

```
con <- dbConnect(RSQLite::SQLite(), "../../datafolder/worklife.sqlite")
```

We now have an empty database which we can fill with the datasets we made above. To add `dataset_salaries` and `dataset_absence`, use `dbCreateTable`. It takes an R dataframe and converts it to a table inside the database. This function takes three arguments: dbCreateTable(conn, name, field):

- Conn: The DBI connection object.
- Name: The name of the new table.
- Field: The data you want to insert. It has to be either a data frame or a character vector.

`dbListTables` will then show the tables in the database.

```
dbWriteTable(con, "salaries", dataset_salaries)
dbWriteTable(con, "absence", dataset_absence)

dbListTables(con)
```

```
## [1] "absence"  "salaries"
```

To add more data to your database, use can use `dbWriteTable`.

```
salaries_datanew <- as_tibble(list(id = c(6, 7),
                                   age = c(31, 73),
                                   occupation = c("doctor", "self-employed"),
                                   salary = c(7100, 1100)))

dbWriteTable(con, name = "salaries", value = salaries_datanew, append = TRUE)
```

And to see what the table contains and/or fetch it into R, use `dbReadTable`.

```
dbReadTable(con, "salaries")
```

```
##   id age    occupation salary
## 1  1  43     carpenter   5300
## 2  2  53        doctor   6800
## 3  3  25    accountant   2100
## 4  4  37        priest   1600
## 5  5  41  reseptionist   1300
## 6  6  31        doctor   7100
## 7  7  73 self-employed   1100
```

```
df <- dbReadTable(con, "salaries")
```

To remove a table from a database, use `dbRemoveTable`.

```
dbRemoveTable(con, "absence")

dbListTables(con)
```

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

To disconnect from the database:

```
dbDisconnect(con)
```

To learn more on how work with SQLite in R, you can visit this page.

SQL

SQL stands for Structured Query Language. It's the computer language used to store, manipulate and retrieve data from relational databases. This is done through "SQL queries", a chuck of code that tells the computer to do something with a database, for example fetch some data from it.

The above examples uses R-code to work with databases, but this is actually just a wrapper around SQL. Behind the curtains, they are not R-queries but SQL-queries. It's generally good to know a little SQL, because then you can work with databases from any program. Here, we offer a very brief introduction.

To work with SQL in R, first load the package `sqldf`.

```
library(sqldf)
```

Now, you can write SQL-code in R.

```
sqldf('SELECT age
       FROM salaries
       WHERE occupation = "doctor"')
```

Of course, this all depends on the kind of database you use again (MySQL, Oracle, PostgreSQL, etc.). If we wanted to use SQL to read from the worklife.sqlite file, we could use the `dbGetQuery` function, add the name of the database-connection object and then the SQL-code. For example, the code below extracts the variable `age` from the table object `salaries` and filters the variable to where `occupation` is "doctor". So we get the ages of the people in the dataset who are doctors.

```
con <- dbConnect(RSQLite::SQLite(), "../../datafolder/worklife.sqlite")

dbGetQuery(con,
           'SELECT age
            FROM salaries
            WHERE occupation = "doctor"')
```

```
##   age
## 1  53
## 2  31
```

```
dbDisconnect(con)
```

To learn more about how to make SQL queries in R, visit this link

Publicly available datasets

There are lots of datasets out there, and many of them are even structured. Some of them might be useful for your projects. The department of political science has a webpage with many relevant datasets sorted by topic.

Other examples include:

- The Nonviolent and Violent Campaigns and Outcomes (NAVCO) Data Project
- Varities of democracy
- Data on armed conflict, PRIO
- European social survey
- Quality of government
- Afrobarometro
- Latinobarometro
- Political party database project (PPDB)
- Parliaments and government database (PARLGOV)

Many researchers also publish data for their papers in Harvard dataverse, and there can be useful avenues for using these data on other things.