Inf2 – Foundations of Data Science 2022

Lecture: Data

25th September 2022

1 Data and metadata

What is data? Since the 17th Century the word **data** has referred to "things known or assumed as facts, and made the basis of reasoning or calculation" (OED). Data according to this definition has existed for 1000s of years, for example in the form of handwritten tables of scientific measurement, census records and financial accounts such as those found in Ancient Egypt (Figure 1). This data is all collected by humans, and is typically written down. The amount of data it is possible to collect is therefore low, and any calculations performed on the data had to be done by hand.

Since the mid-20th century, the word has also referred to "the quantities, characters or symbols on which operations are performed by computers and other automatic equipment, and which may be stored and transmitted in the form of electrical signals, records or magnetic, optical or mechanical recording media" (OED). Thus, the definition of data has expanded. Digital data comes in many forms, for example images, sound files, emails and documents, scientific databases and medical records. Some of these forms of data are collected by machine or automatically – for example a digital camera sets creates the set of numbers that describe an image; a web server records the details of the every visit to a web page. The amount of data it has been possible to collect has increased massively, as has our ability to process that data.

Nevertheless, in this age of big data, humans are still collecting smaller datasets. For example opinion pollsters might survey 1000 people in a poll, or ecologists might measure the height and weight of a sample of 100 squirrels.

Structured and unstructured data There is an important distinction is between structured data and **unstructured data**. In **structured data**, the data is organised according to a clear structure. An example of structured data is a table describing attributes of Star Wars characters with columns such as name, "height", "weight" and eye colour. Structured data needn't be a table: for example a bibliographic database, with different types of entries for books and academic articles. It is easy to query structured data: for example find all characters in Star Wars with blue eyes.

Unstructured data does not have a predefined structure. Text and images are common examples of unstructured data. It is typically much harder to extract information from unstructured data than from structured data. For example in a block of text written about Star Wars characters, would find it hard to identify all characters with blue eyes.

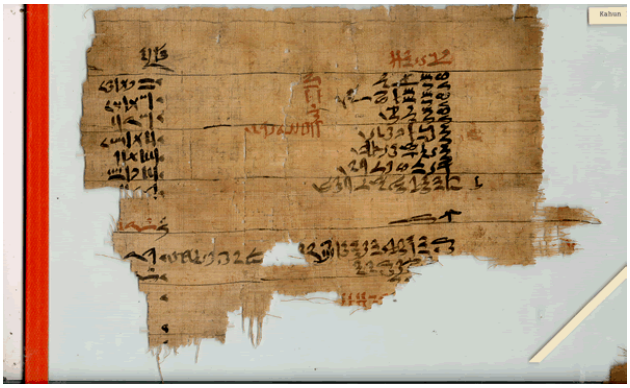

Figure 1: Ancient Egyptian grain account from between 2025–1700 BC found in the Lahun Papyri. Acquisition UC32189, Digital Egypt for Universities, University College London, `https://www.ucl.ac.uk/museums-static/digitalegypt/lahun/papyri.html`.

Table 1: Characteristics of some imaginary squirrels.

Name	Weight (g)	Length (mm)	Sex	Age
Jakub	320	211.0	Male	Under 1 year
Fiona	342	222.0	Female	1–2 years
Cameron	330	215.0	Male	2+ years

Metadata We contain terabytes of data files, but they are useless if we know what they *mean*. **Metadata** – literally "about data" – is information describing the data files it accompanies. The metadata may be a description in a text file (often called README) or it may be in a structured format. Whenever a dataset is created, stored and shared, the data meaning of the data should be recorded. It is surprising how often the meaning of datasets is not clearly described. The metadata should describe how the data were collected, including if there were any legal or ethical considerations, the format of the data files and the meaning of variables in the data files.

2 Tabular data and variables

Tabular data In this course we focus on a common class of structured data, **tabular data**. Tabular data is data arranged in a table. The meaning of each cell in the table is determined by the column and/or row labels. Table 1 shows an example of tabular data.

Tidy Data Tidy data is a subset of tabular data (Wickham, 2014). In a tidy dataset, each column corresponds to a **variable** (or **attribute**) and each row corresponds to an instance possessing each of the variables.

For example, suppose we capture squirrels and measure their weight (in grams) and length (in millimetres), and note down their sex. The variables represented in the columns would be "Name" (assuming we've named the squirrels we are observing), "Weight (g)", "Length (mm)"

and "Sex". Each row contains the value of these variables.

Typically, tidy data is **multivariate**, i.e. it has multiple variables. In the special case of two variables we refer to it as **bivariate**, and when we consider only one variable, we call it **univariate**.

Messy data We call data that is not in the tidy format messy. There are a number of ways in which data can be messy, for example by using variables in as column headings. Messy data may actually be easier for the human eye to comprehend, but is not so easy to manipulate using software tools, so for the moment we will assume we have tidy data.

Types of variables Variables can be of various types:

- **Numerical variables** are quantities that can be measured (continuous) or counted (discrete). For example, weight and length are continuous numeric variables. In contrast, the number of babies a squirrel gives birth to in a year is a discrete variable, since we cannot have a fractional number of babies. Continuous variables often have a physical dimension (e.g. weight or length) and it is important to quote them with their unit.
- **Categorical variables** can take on one of a number of values. For example, the sex of the squirrel is a categorical variable since it can be "Male" or "Female".
- **Ordinal variables** can take on one of a number of categories, but those categories are ordered. For example, we might estimate the age of a squirrel as "Under 1 year old", "1 to 2 years old" or "More than 2 years old". There are only three categories, but we can order them from youngest to oldest.
- **String variables** contain string information such as names or a comment from a survey form.

3 Working with tabular data

In the week 1 lab, you will learn how to read stored tabular data, select rows and columns, and filter tabular data.

Reading stored data Tabular data can be stored in a variety of formats, and Data Science packages (for example Pandas or R) have functions to read in this data:

- Text file: common formats are comma-separated variable (CSV), tab separated variable (TSV). There are various standards for the precise formatting of the files, for example if the data enclosed by cells are enclosed by quotes. Sometimes there are a few lines of metadata at the top of the file. Data science packages functions to read in text files, have many options, for example allowing you to skip lines at the head of the file, or deal with separators other than tabs or commas.

- Binary file: common formats are the Excel spreadsheets or Open Document Format spreadsheets. The modern versions of both of these are in fact ZIP files containing an XML file with the spreadsheet information, and any other files (e.g. embedded images).
- Databases: for example MySQL or SQLlite. Here there is a database server, and data is extracted by running SQL (Structured Query Language).[1]

Text files have the advantage of being human-readable, and also enforce a discipline of encoding information solely in the content of the cells – it is not possible to encode information by colour-coding cells, as it is in spreadsheets. Conversely, spreadsheets have the advantage be being format-able, which can help with readability.

Selecting rows and columns Data science packages have methods for extracting rows and columns from tables. With tidy data, extracting a row selects all the data connected with one observation. Extracting a column selects every instance of one variable.

Filtering data We call finding a subset of the data based on the value of some variable **filtering**. For example, we may wish to filter out all the squirrels longer than 220 mm.

4 Data wrangling

Before you can undertake data science analyses, you need to get (or "wrangle") the data into a suitable form, a process that is called **data wrangling**. In the week 2 lab you will learn about cleaning data, dealing with missing data and joining datasets.

Cleaning data The quality of any analysis of data can only be as good as the data itself – and the data itself may have many types of problems:

- Data entry: for example, if we have collected data in a free-text survey, respondents may not have entered a time in a uniform format (e.g. "16:00", or "4pm", or "4" or "16" or "17 (CET)" may all mean the same thing. Or someone may have typed "08" when they meant "80".
- Mixing text and numbers: for example we might have recorded "210g" or "0.21kg" in the weight cell for our squirrels.
- Missing data: perhaps a sensor wasn't working for a few minutes or hours, so some readings are missing. Does it record "0" in this situation? Or "−1". The metadata should tell us, but maybe it doesn't...

[1] You can learn how process and analyse data using SQL in the 3rd year course http://www.drps.ed.ac.uk/22-23/dpt/cxinfr10080.htm.

- Mislabelled data: A column may have been mislabelled. For example, we might be recording the temperature of heating water entering and leaving the Informatics Forum, and the temperature of heating water entering and leaving Dugald Stewart Building (DSB). To get a measure how much heat the Forum is using we subtract the temperature of the water leaving from the water entering, and the same for DSB. But if we subtract the temperature of water leaving DSB from water entering the Forum, we will get nonsense. (This scenario actually happened.)
- Duplicated data: perhaps someone has made a copy-and-paste error in a spreadsheet.
- Faulty sensor: perhaps a sensor goes wrong, and starts giving values that are implausible.

In summary, there are many potential problems with data: **the data is out to get you!**. One of the most important jobs of a data scientist is to check the data quality, and fix problems. You need to approach the data in the spirit of critical evaluation: Is this value reasonable? Is that pattern strange? Does the data look too good?

We use the term **data cleaning** to describe the process of checking and fixing problems in data. You should start data checking and cleaning after loading the dataset, but problems with data also emerge in the process of visualisation, which we will come to later.

Data cleaning is very time-consuming: it is commonly said that 80% of the time of a data science project is spent on cleaning (Dasu and Johnson, 2003) though some more recent estimates suggest time spent cleaning is less than 30% (Anaconda, 2020), though the same report also estimates loading and visualising data take around 20%. In any event, data cleaning can take a long time, needs to be done carefully, and can be quite fiddly and frustrating.

Missing data Modern data science packages have a special values – `NA` or `NaN` (**Not applicable** or **Not a Number**) – to describe data that is missing. It's very helpful to ensure that data that you regard as missing shows as `NA` or `NaN` rather than as a default value (e.g. 0 or -1), as this can help with filtering out missing data.

Pandas is somewhat confusing in its handling of missing data. Strictly speaking `NA` applies to missing string, categorical or numeric data, whereas `NaN` applies only to missing numbers. However, Pandas represents missing data as `NaN`, but functions to check for missing data refer to `NA`, e.g. `.isna()`.

Once missing data is identified, you have to decide what to do with it, which may depend on the analysis you are undertaking. Sometimes it can be possible to keep observations that contain missing data, but some analyses only work if every variable in every observation has a value. In this case you may need to drop any rows containing `NaN`.

Merging tabular data using database-style joins Often we wish to combine data from two sources that relates to the same entities. For example, we might have recorded the times that some of our squirrels could undertake an obstacle course. Now, suppose we wanted to compare the times of completion to the characteristics of the squirrel (Weight, Length, Sex and Age). Both Table 1 and Table 2 share a common variable: "Name". We can match up the

Table 2: Time taken to complete obstacle course by squirrels

Name	Date	Time (s)
Fiona	2021-05-06	67.5
Fiona	2021-05-10	50.2
Cameron	2021-05-08	55.6
Lily	2022-07-13	45.0

rows of both tables by using the `merge` function in Pandas with "Name" as the **key** that binds the tables. In relational database terminology the equivalent operation is called a **join**.

Notice that not all the squirrels in the Table 1 undertook the obstacle course, and there is one squirrel ("Lily") who is present in Table 2 but not in the first table.

There are various ways of merging, which deal with this mismatch:

- **inner join** (Table 3): Only the squirrels in both datasets will be present in the joined dataset. Note that the key Fiona in Table 1 matches two rows in Table 2, so there are two corresponding rows in the joined table. The data describing Fiona's characteristics is repeated in these rows, but the unique information about the data and time of the obstacle course run are not repeated. Jakub isn't present in this table since Jakub hasn't had a time recorded on the obstacle course in Table 2.
- **left join** (Table 4): All squirrels present in the first (left) table passed to the merge function will be retained in the merged table. When the key isn't present in the second table we fill in the missing values with `NaN`. In Table 4 Jakub has `NaN` values for data and time, since Jakub hasn't had a time recorded on the obstacle course in Table 2.
- **outer join** (Table 5): All squirrels in both datasets will be present in the joined dataset. When the key isn't present in the first or second table we fill in the missing values with `NaN`. In Table 5 Jakub and Lily are present because they are present either in Table 1 (Jakub) or Table 2 (Lily). Jakub has `NaN` values for data and time, since Jakub hasn't had a time recorded on the obstacle course, and Lily's age, height, weight etc. are missing, since Lily didn't appear in Table 1.

References

Anaconda (2020). 'The state of data science 2020: Moving from hype toward maturity'. URL `https://www.anaconda.com/state-of-data-science-2020`

Dasu, T. and Johnson, T. (2003). *Exploratory data mining and data cleaning*. Wiley

Wickham, H. (2014). 'Tidy data'. *Journal of Statistical Software* **59**. URL `https://www.jstatsoft.org/index.php/jss/article/view/v059i10/v59i10.pdf`

Table 3: Results of inner join applied to Tables 1 and 2.

Name	Weight (g)	Length (mm)	Sex	Age	Date	Time (s)
Fiona	342	222.0	Female	1-2 years	2021-05-06	67.5
Fiona	342	222.0	Female	1-2 years	2021-05-10	50.2
Cameron	330	215.0	Male	2+ years	2021-05-08	55.6

Table 4: Results of left join applied to Tables 1 and 2.

Name	Weight (g)	Length (mm)	Sex	Age	Date	Time (s)
Jakub	320	211.0	Male	Under 1 year	nan	nan
Fiona	342	222.0	Female	1-2 years	2021-05-06	67.5
Fiona	342	222.0	Female	1-2 years	2021-05-10	50.2
Cameron	330	215.0	Male	2+ years	2021-05-08	55.6

Table 5: Results of outer join applied to Tables 1 and 2.

Name	Weight (g)	Length (mm)	Sex	Age	Date	Time (s)
Jakub	320.0	211.0	Male	Under 1 year	nan	nan
Fiona	342.0	222.0	Female	1-2 years	2021-05-06	67.5
Fiona	342.0	222.0	Female	1-2 years	2021-05-10	50.2
Cameron	330.0	215.0	Male	2+ years	2021-05-08	55.6
Lily	nan	nan	nan	nan	2022-07-13	45.0