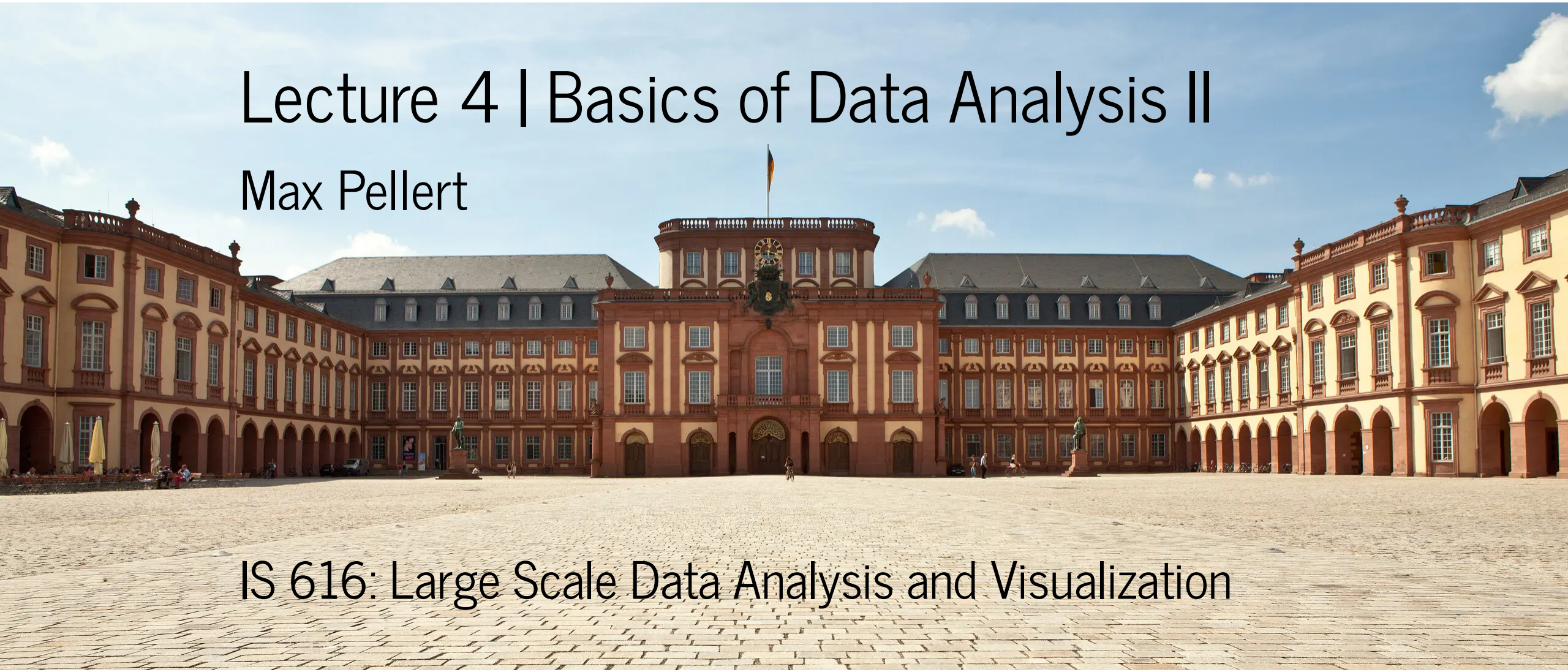
# Lecture 4 | Basics of Data Analysis II

Max Pellert



## IS 616: Large Scale Data Analysis and Visualization

Tidy data and tidy tools make data analysis easier…

…by easing the transitions between manipulation, visualization and modeling

Tidy data and tidy tools make data analysis easier…

…by easing the transitions between manipulation, **visualization** and modeling.

“The tidy data standard has been designed to facilitate initial exploration and analysis of the data, and to simplify the development of data analysis tools that work well together.

Current tools often require translation. You have to spend time munging the output from one tool so you can input it into another.

Tidy datasets and tidy tools work hand in hand to make data analysis easier, allowing you to focus on the interesting domain problem, not on the uninteresting logistics of data.”

Wickham, H. (2014). Tidy Data. Journal of Statistical Software, 59(10). https://doi.org/10.18637/jss.v059.i10

# Definition

“Tidy data sets are all alike but every messy dataset is messy in its own way”

Tidy data sets provide a standardized way to link

the structure of a data set (its physical layout)

with its semantics (its meaning)

# A special vocabulary

is used to describe the structure and the semantics of data sets

We will try to use this terms in the rest of the course when talking about data

The definitions of the vocabulary will us allow to define tidy data

# Data structure

Statistical data sets are:

rectangular tables

that are made up of **rows** and **columns**

columns are always labelled

rows are sometimes labelled

Question: Is it enough just to describe structure of data sets?

| | treatmenta | treatmentb |
|---|---|---|
| John Smith | — | 2 |
| Jane Doe | 16 | 11 |
| Mary Johnson | 3 | 1 |

Table 1: Typical presentation dataset.

| | John Smith | Jane Doe | Mary Johnson |
|---|---|---|---|
| treatmenta | — | 16 | 3 |
| treatmentb | 2 | 11 | 1 |

Table 2: The same data as in Table 1 but structured differently.

Simple transpose of the same data, i.e. rows to columns and columns to rows, cannot be adequately described using just a rows and columns terminology

Different layout but both tables are actually representations of the same data

–> We also need to describe the underlying semantics of the data

# Data Semantics

A data set is a collection of **values**, usually either numbers (if "quantitative") or strings (if "qualitative")

Values are organized in two ways

Every value belongs to a **variable**

and an **observation**

A variable contains all values that measure the same underlying attribute (like height or temperature or duration) across units (like a person or a day or a race)

An observation contains all values measured on the same unit (like a person or a day or a race) across attributes

Table 3 reorganizes Table 1 to make the values, variables and observations more clear. The dataset contains 18 values representing three variables and six observations. The variables are:

1. `person`, with three possible values (John Smith, Mary Johnson, and Jane Doe).
2. `treatment`, with two possible values (a and b).
3. `result`, with five or six values depending on how you think of the missing value (—, 16, 3, 2, 11, 1).

| person | treatment | result |
|---|---|---|
| John Smith | a | — |
| Jane Doe | a | 16 |
| Mary Johnson | a | 3 |
| John Smith | b | 2 |
| Jane Doe | b | 11 |
| Mary Johnson | b | 1 |

Table 3: The same data as in Table 1 but with variables in columns and observations in rows.

The experimental design (or more general “the way we retrieved the data”) informs us about the structure of the observations

For example, we assumed a completely crossed design where every treatment was to be measured for each person

Here, the design says that each possible value should have been recorded, which we make explicit by adding a missing value

Depending on the design, missing values can also be just dropped

But, the absence of evidence doesn’t mean the evidence of absence! Only in very special circumstances it could be OK to replace NA by 0 for example

Following the publication of this Letter, Beheim and colleagues submitted a Matters Arising in which they argued that our primary results were called into question by our treatment of missing data[1]. In our research, we attempted to test the 'big gods' hypothesis even-handedly using the best available evidence, and we made our data and code available during the review process and after publication, in line with best practice in open science. Nevertheless, we accept that we should have labelled moralizing gods as 'absent' or 'inferred absent' rather than 'unknown' in portions of our dataset before the dates of the first appearance, rather than converting 'NAs' to zeros during the phase of analysis. Since this Letter was published, we have thoroughly refined our data and analyses, and have found that our original conclusions are still strongly supported[2,3]. However, the differences between our revised analyses and the original Letter are substantial enough to warrant a Retraction of the original Letter. We have submitted the enhanced analyses for peer review and potential publication in another journal. We encourage the community to refer to these new papers in future instead of this now-retracted Letter. We apologize to the scientific community for the unintended confusion. The authors John Baines, Alan Covey and Kevin Feeney do not agree with this Retraction.

https://www.nature.com/articles/s41586-019-1043-4

It’s actually hard to define variables and observations precisely in general

But, usually it’s quite easy to figure out what are variables and what are observations for a given data set (although there can be some room for interpretation)

For example, two columns of “height” and “weight” would easily be called variables

But, what about “height” and “width”?

Those could also values of a “dimension” variable

Similarly, columns “home phone” and “office phone” can be thought of as variables

But for example in the context of fraud detection, we could think of variable “phone number” and “phone type” instead

This would for example allow to inspect duplicate numbers more easily, as individuals using the same phone number would be suspicious

**Thinking in this way how to prepare the data in the best way for the type of problem that you face is one of the most important skills in data analysis!**

# Some rules of thumb

It is usually easier to describe functional relationships between variables

For example, “density” as the ratio of “weight” to “volume”

It is usually easier to make comparisons between groups of observations than by groups of variables

For example average of group a vs average of group b or by creating multiple subplots of the same variables for different groups (small multiples)

In a given analysis, there may be multiple levels of observations. For example, in a trial of new allergy medication we might have three observational types: demographic data collected from each person (`age`, `sex`, `race`), medical data collected from each person on each day (`number of sneezes`, `redness of eyes`), and meteorological data collected on each day (`temperature`, `pollen count`).

# What makes data tidy?

1. Each variable forms a column.
2. Each observation forms a row.
3. Each type of observational unit forms a table.

While the order of variables and observations does not affect analysis, a good ordering makes it easier to scan the raw values. One way of organizing variables is by their role in the analysis: are values fixed by the design of the data collection, or are they measured during the course of the experiment? Fixed variables describe the experimental design and are known in advance. Computer scientists often call fixed variables dimensions, and statisticians usually denote them with subscripts on random variables. Measured variables are what we actually measure in the study. Fixed variables should come first, followed by measured variables, each ordered so that related variables are contiguous. Rows can then be ordered by the first variable, breaking ties with the second and subsequent (fixed) variables. This is the convention adopted by all tabular displays in this paper.

# Tidying messy datasets

- Column headers are values, not variable names.
- Multiple variables are stored in one column.
- Variables are stored in both rows and columns.
- Multiple types of observational units are stored in the same table.
- A single observational unit is stored in multiple tables.

https://github.com/hadley/tidy-data

# Column headers are values not variable names

| religion | <$10k | $10–20k | $20–30k | $30–40k | $40–50k | $50–75k |
|---|---|---|---|---|---|---|
| Agnostic | 27 | 34 | 60 | 81 | 76 | 137 |
| Atheist | 12 | 27 | 37 | 52 | 35 | 70 |
| Buddhist | 27 | 21 | 30 | 34 | 33 | 58 |
| Catholic | 418 | 617 | 732 | 670 | 638 | 1116 |
| Don't know/refused | 15 | 14 | 15 | 11 | 10 | 35 |
| Evangelical Prot | 575 | 869 | 1064 | 982 | 881 | 1486 |
| Hindu | 1 | 9 | 7 | 9 | 11 | 34 |
| Historically Black Prot | 228 | 244 | 236 | 238 | 197 | 223 |
| Jehovah's Witness | 20 | 27 | 24 | 24 | 21 | 30 |
| Jewish | 19 | 19 | 25 | 25 | 30 | 95 |

Table 4: The first ten rows of data on income and religion from the Pew Forum. Three columns, $75–100k, $100–150k and >150k, have been omitted.

| religion | income | freq |
|---|---|---|
| Agnostic | <$10k | 27 |
| Agnostic | $10–20k | 34 |
| Agnostic | $20–30k | 60 |
| Agnostic | $30–40k | 81 |
| Agnostic | $40–50k | 76 |
| Agnostic | $50–75k | 137 |
| Agnostic | $75–100k | 122 |
| Agnostic | $100–150k | 109 |
| Agnostic | >150k | 84 |
| Agnostic | Don't know/refused | 96 |

| row | a | b | c |
|---|---|---|---|
| A | 1 | 4 | 7 |
| B | 2 | 5 | 8 |
| C | 3 | 6 | 9 |

(a) Raw data

| row | column | value |
|---|---|---|
| A | a | 1 |
| B | a | 2 |
| C | a | 3 |
| A | b | 4 |
| B | b | 5 |
| C | b | 6 |
| A | c | 7 |
| B | c | 8 |
| C | c | 9 |

(b) Molten data

# Multiple variables are stored in one column

| country | year | m014 | m1524 | m2534 | m3544 | m4554 | m5564 | m65 | mu | f014 |
|---|---|---|---|---|---|---|---|---|---|---|
| AD | 2000 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | — | — |
| AE | 2000 | 2 | 4 | 4 | 6 | 5 | 12 | 10 | — | 3 |
| AF | 2000 | 52 | 228 | 183 | 149 | 129 | 94 | 80 | — | 93 |
| AG | 2000 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | — | 1 |
| AL | 2000 | 2 | 19 | 21 | 14 | 24 | 19 | 16 | — | 3 |
| AM | 2000 | 2 | 152 | 130 | 131 | 63 | 26 | 21 | — | 1 |
| AN | 2000 | 0 | 0 | 1 | 2 | 0 | 0 | 0 | — | 0 |
| AO | 2000 | 186 | 999 | 1003 | 912 | 482 | 312 | 194 | — | 247 |
| AR | 2000 | 97 | 278 | 594 | 402 | 419 | 368 | 330 | — | 121 |
| AS | 2000 | — | — | — | — | 1 | 1 | — | — | — |

Table 9: Original TB dataset. Corresponding to each 'm' column for males, there is also an 'f' column for females, `f1524`, `f2534` and so on. These are not shown to conserve space. Note the mixture of 0s and missing values (—). This is due to the data collection process and the distinction is important for this dataset.

| country | year | column | cases |
|---|---|---|---|
| AD | 2000 | m014 | 0 |
| AD | 2000 | m1524 | 0 |
| AD | 2000 | m2534 | 1 |
| AD | 2000 | m3544 | 0 |
| AD | 2000 | m4554 | 0 |
| AD | 2000 | m5564 | 0 |
| AD | 2000 | m65 | 0 |
| AE | 2000 | m014 | 2 |
| AE | 2000 | m1524 | 4 |
| AE | 2000 | m2534 | 4 |
| AE | 2000 | m3544 | 6 |
| AE | 2000 | m4554 | 5 |
| AE | 2000 | m5564 | 12 |
| AE | 2000 | m65 | 10 |
| AE | 2000 | f014 | 3 |

(a) Molten data

| country | year | sex | age | cases |
|---|---|---|---|---|
| AD | 2000 | m | 0–14 | 0 |
| AD | 2000 | m | 15–24 | 0 |
| AD | 2000 | m | 25–34 | 1 |
| AD | 2000 | m | 35–44 | 0 |
| AD | 2000 | m | 45–54 | 0 |
| AD | 2000 | m | 55–64 | 0 |
| AD | 2000 | m | 65+ | 0 |
| AE | 2000 | m | 0–14 | 2 |
| AE | 2000 | m | 15–24 | 4 |
| AE | 2000 | m | 25–34 | 4 |
| AE | 2000 | m | 35–44 | 6 |
| AE | 2000 | m | 45–54 | 5 |
| AE | 2000 | m | 55–64 | 12 |
| AE | 2000 | m | 65+ | 10 |
| AE | 2000 | f | 0-14 | 3 |

(b) Tidy data

# Variables are stored in both rows and columns

| id | year | month | element | d1 | d2 | d3 | d4 | d5 | d6 | d7 | d8 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| MX17004 | 2010 | 1 | tmax | — | — | — | — | — | — | — | — |
| MX17004 | 2010 | 1 | tmin | — | — | — | — | — | — | — | — |
| MX17004 | 2010 | 2 | tmax | — | 27.3 | 24.1 | — | — | — | — | — |
| MX17004 | 2010 | 2 | tmin | — | 14.4 | 14.4 | — | — | — | — | — |
| MX17004 | 2010 | 3 | tmax | — | — | — | — | 32.1 | — | — | — |
| MX17004 | 2010 | 3 | tmin | — | — | — | — | 14.2 | — | — | — |
| MX17004 | 2010 | 4 | tmax | — | — | — | — | — | — | — | — |
| MX17004 | 2010 | 4 | tmin | — | — | — | — | — | — | — | — |
| MX17004 | 2010 | 5 | tmax | — | — | — | — | — | — | — | — |
| MX17004 | 2010 | 5 | tmin | — | — | — | — | — | — | — | — |

Table 11: Original weather dataset. There is a column for each possible day in the month. Columns d9 to d31 have been omitted to conserve space.

| id | date | element | value |
|---|---|---|---|
| MX17004 | 2010-01-30 | tmax | 27.8 |
| MX17004 | 2010-01-30 | tmin | 14.5 |
| MX17004 | 2010-02-02 | tmax | 27.3 |
| MX17004 | 2010-02-02 | tmin | 14.4 |
| MX17004 | 2010-02-03 | tmax | 24.1 |
| MX17004 | 2010-02-03 | tmin | 14.4 |
| MX17004 | 2010-02-11 | tmax | 29.7 |
| MX17004 | 2010-02-11 | tmin | 13.4 |
| MX17004 | 2010-02-23 | tmax | 29.9 |
| MX17004 | 2010-02-23 | tmin | 10.7 |

(a) Molten data

| id | date | tmax | tmin |
|---|---|---|---|
| MX17004 | 2010-01-30 | 27.8 | 14.5 |
| MX17004 | 2010-02-02 | 27.3 | 14.4 |
| MX17004 | 2010-02-03 | 24.1 | 14.4 |
| MX17004 | 2010-02-11 | 29.7 | 13.4 |
| MX17004 | 2010-02-23 | 29.9 | 10.7 |
| MX17004 | 2010-03-05 | 32.1 | 14.2 |
| MX17004 | 2010-03-10 | 34.5 | 16.8 |
| MX17004 | 2010-03-16 | 31.1 | 17.6 |
| MX17004 | 2010-04-27 | 36.3 | 16.7 |
| MX17004 | 2010-05-27 | 33.2 | 18.2 |

(b) Tidy data

Table 12: (a) Molten weather dataset. This is almost tidy, but instead of values, the `element` column contains names of variables. Missing values are dropped to conserve space. (b) Tidy weather dataset. Each row represents the meteorological measurements for a single day. There are two measured variables, minimum (`tmin`) and maximum (`tmax`) temperature; all other variables are fixed.

# Multiple types in one table

| id | artist | track | time |
|---|---|---|---|
| 1 | 2 Pac | Baby Don't Cry | 4:22 |
| 2 | 2Ge+her | The Hardest Part Of ... | 3:15 |
| 3 | 3 Doors Down | Kryptonite | 3:53 |
| 4 | 3 Doors Down | Loser | 4:24 |
| 5 | 504 Boyz | Wobble Wobble | 3:35 |
| 6 | 98ˆ0 | Give Me Just One Nig... | 3:24 |
| 7 | A*Teens | Dancing Queen | 3:44 |
| 8 | Aaliyah | I Don't Wanna | 4:15 |
| 9 | Aaliyah | Try Again | 4:03 |
| 10 | Adams, Yolanda | Open My Heart | 5:30 |
| 11 | Adkins, Trace | More | 3:05 |
| 12 | Aguilera, Christina | Come On Over Baby | 3:38 |
| 13 | Aguilera, Christina | I Turn To You | 4:00 |
| 14 | Aguilera, Christina | What A Girl Wants | 3:18 |
| 15 | Alice Deejay | Better Off Alone | 6:50 |

| id | date | rank |
|---|---|---|
| 1 | 2000-02-26 | 87 |
| 1 | 2000-03-04 | 82 |
| 1 | 2000-03-11 | 72 |
| 1 | 2000-03-18 | 77 |
| 1 | 2000-03-25 | 87 |
| 1 | 2000-04-01 | 94 |
| 1 | 2000-04-08 | 99 |
| 2 | 2000-09-02 | 91 |
| 2 | 2000-09-09 | 87 |
| 2 | 2000-09-16 | 92 |
| 3 | 2000-04-08 | 81 |
| 3 | 2000-04-15 | 70 |
| 3 | 2000-04-22 | 68 |
| 3 | 2000-04-29 | 67 |
| 3 | 2000-05-06 | 66 |

Table 13: Normalized Billboard dataset split up into song dataset (left) and rank dataset (right). First 15 rows of each dataset shown; `genre` omitted from song dataset, `week` omitted from rank dataset.

# One type in multiple tables

"It is also common to find data values about a single type of observational unit spread out over multiple tables or files.

These tables and files are often split up by another variable, so that each represents a single year, person, or location."

1. Read the files into a list of tables
2. For each table, add a new column that records the original file name (because the file name is often the value of an important variable)
3. Combine all tables into a single table

Can be achieved in multiple ways, for example along the lines of

```r
library(data.table)

together_dt <- rbindlist(lapply(paste0("filename_",1:10),function(x) fre
```

```python
import pandas as pd

together_df = pd.concat((pd.read_csv(f) for f in all_files), ignore_inde
```

and in many other ways

Tidy visualization tools only need to be input-tidy as their output is visual

Domain specific languages work particularly well for the visualization of tidy datasets because they can describe a visualization as a mapping between variables and aesthetic properties of the graph (e.g., position, size, shape and color)

This is the idea behind the grammar of graphics (Wilkinson 2005), and the layered grammar of graphics (Wickham 2010), an extension tailored specifically for R

ggplot2 was also ported to Python with the package "plotnine"

# One more command line tool


jq is a lightweight and flexible command-line JSON processor.

Download jq 1.7 Try online at jqplay.org!

jq is like `sed` for JSON data - you can use it to slice and filter and map and transform structured data with the same ease that `sed`, `awk`, `grep` and friends let you play with text.

https://jqlang.github.io/

https://jqlang.github.io/jq/tutorial/

GitHub has a JSON API, so let's play with that. This URL gets us the last 5 commits from the jq repo.

```
curl -s 'https://api.github.com/repos/jqlang/jq/commits?per_page=5'
```

```
## [
##   {
##     "sha": "8f81668014f4df2654aa9ab674b5498aa9446441",
##     "node_id": "C_kwDOAE3WVdoAKDhmODE2NjgwMTRmNGRmMjY1NGFhOWFiNjc0YjU
##     "commit": {
##       "author": {
##         "name": "taoky",
##         "email": "taoky99@outlook.com",
##         "date": "2023-09-22T00:18:41Z"
##       },
##       "committer": {
##         "name": "GitHub",
##         "email": "noreply@github.com",
##         "date": "2023-09-22T00:18:41Z"
##       },
##       "message": "Fix the default colors to use 39, the default foreg
##       "tree": {
```


**command line tool and library**
for transferring data with URLs
(since 1998)

# GNU Wget

GNU Wget is a free software package for retrieving files using HTTP, HTTPS, FTP and FTPS, the most widely used Internet protocols. It is a non-interactive commandline tool, so it may easily be called from scripts, `cron` jobs, terminals without X-Windows support, etc.

GitHub returns nicely formatted JSON. For servers that don’t, it can be helpful to pipe the response through jq to pretty-print it. The simplest jq program is the expression “.”, which takes the input and produces it unchanged as output.

```
curl -s 'https://api.github.com/repos/jqlang/jq/commits?per_page=5' | jq
```

```
## [
##   {
##     "sha": "8f81668014f4df2654aa9ab674b5498aa9446441",
##     "node_id": "C_kwDOAE3WVdoAKDhmODE2NjgwMTRmNGRmMjY1NGFhOWFiNjc0YjU
##     "commit": {
##       "author": {
##         "name": "taoky",
##         "email": "taoky99@outlook.com",
##         "date": "2023-09-22T00:18:41Z"
##       },
##       "committer": {
##         "name": "GitHub",
##         "email": "noreply@github.com",
##         "date": "2023-09-22T00:18:41Z"
##       },
##       "message": "Fix the default colors to use 39, the default foreg
##       "tree": {
```

# We can use jq to extract just the first commit.

```
curl -s 'https://api.github.com/repos/jqlang/jq/commits?per_page=5' | jq
```

```
## {
##   "sha": "8f81668014f4df2654aa9ab674b5498aa9446441",
##   "node_id": "C_kwDOAE3WVdoAKDhmODE2NjgwMTRmNGRmMjY1NGFhOWFiNjc0YjU0O
##   "commit": {
##     "author": {
##       "name": "taoky",
##       "email": "taoky99@outlook.com",
##       "date": "2023-09-22T00:18:41Z"
##     },
##     "committer": {
##       "name": "GitHub",
##       "email": "noreply@github.com",
##       "date": "2023-09-22T00:18:41Z"
##     },
##     "message": "Fix the default colors to use 39, the default foregro
##     "tree": {
##       "sha": "223b8e8db917ea45390102bfff4c9b34b00e2563",
```

There's a lot of info we don't care about there, so we'll restrict it down to the most interesting fields.

```
curl -s 'https://api.github.com/repos/jqlang/jq/commits?per_page=5' | jq
```

```
## {
##   "message": "Fix the default colors to use 39, the default foregrour
##   "name": "GitHub"
## }
```

The | operator in jq feeds the output of one filter (.[0] which gets the first element of the array in the response) into the input of another ({…} which builds an object out of those fields). You can access nested attributes, such as .commit.message.

# Now let's get the rest of the commits (not only the first one).

```
curl -s 'https://api.github.com/repos/jqlang/jq/commits?per_page=5' | jq
```

```
## {
##   "message": "Fix the default colors to use 39, the default foregrour
##   "name": "GitHub"
## }
## {
##   "message": "Bump docker/setup-qemu-action from 2 to 3 (#2900)\n\nBu
##   "name": "GitHub"
## }
## {
##   "message": "Bump docker/setup-buildx-action from 2 to 3 (#2901)\n\r
##   "name": "GitHub"
## }
## {
##   "message": "Bump actions/checkout from 1 to 4 (#2902)\n\nBumps [act
##   "name": "GitHub"
## }
## {
```

.[] returns each element of the array returned in the response, one at a time, which are all fed into {message: .commit.message, name: .commit.committer.name}.

Data in jq is represented as streams of JSON values - every jq expression runs for each value in its input stream, and can produce any number of values to its output stream.