

The Shape of Data

Ken Benoit

ME314 Lecture 2, 22 June 2021

Plan today

- The nature of data
- Units of data
- Data types
- More about git
- Representing text data: Unicode
- Representing dates
- Representing sparse matrix formats
- Data and datasets
 - "tidy" data
 - reshaping data
 - normalization forms
- Lab preview

Data is Fundamental

"You can have data without information, but you cannot have information without data." – Daniel Keys Moran, an American computer programmer and science fiction writer.

"In God we trust. All others must bring data." – W. Edwards Deming, statistician, professor, author, lecturer, and consultant.

Structured data: Index cards

- Origins in the 19th century, with botanist **Carl Linnaeus**, who needed to record species that he was studying
- This was a form of *database*
 - each piece of information about a species formed a *field*
 - each species' entry in the system formed a *record*
 - the records were *indexed* using some reference system

Heyday: Use in libraries to catalog books

a record looked like this

Dewey decimal system

- a proprietary library classification system first published in the United States by Melvil Dewey in 1876
- scheme is made up of ten classes, each divided into ten divisions, each having ten sections
- the system's notation uses Arabic numbers, with three whole numbers making up the main classes and sub-classes and decimals creating further divisions
- Example:

500 Natural sciences and mathematics

 510 Mathematics

 516 Geometry

 516.3 Analytic geometries

 516.37 Metric differential geometries

 516.375 Finsler Geometry

How to index?

- Problem: Could only sort the cards one way
- Re-referencing was literally a manual operation
- Contrast with the idea of electronic indexes, where assets are stored once, and many indexing and reference systems can be applied

Modern database manager

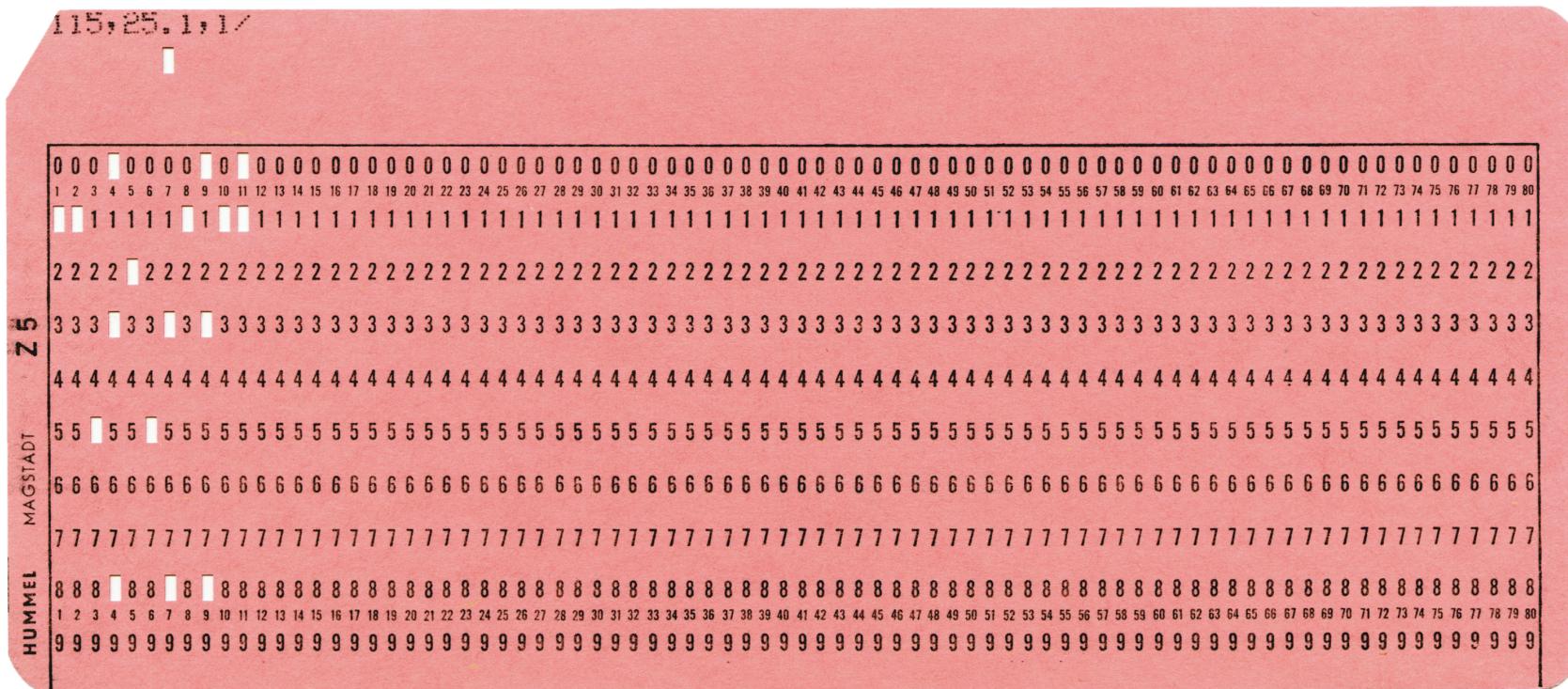
- "Normalizes" data into relational tables, linked by "keys" (more on this in [Week 3](#))

School Table	
ID	Name
S001	University of Technology
S002	University of Applied Science

Student Table			
School ID	ID	Name	DOB
S001	UT-1000	Tommy	05/06/1995
S001	UT-1000	Better	16/04/1995
S002	UAS-1000	Linda	02/09/1995
S002	UAS-1000	Jonathan	22/06/1995

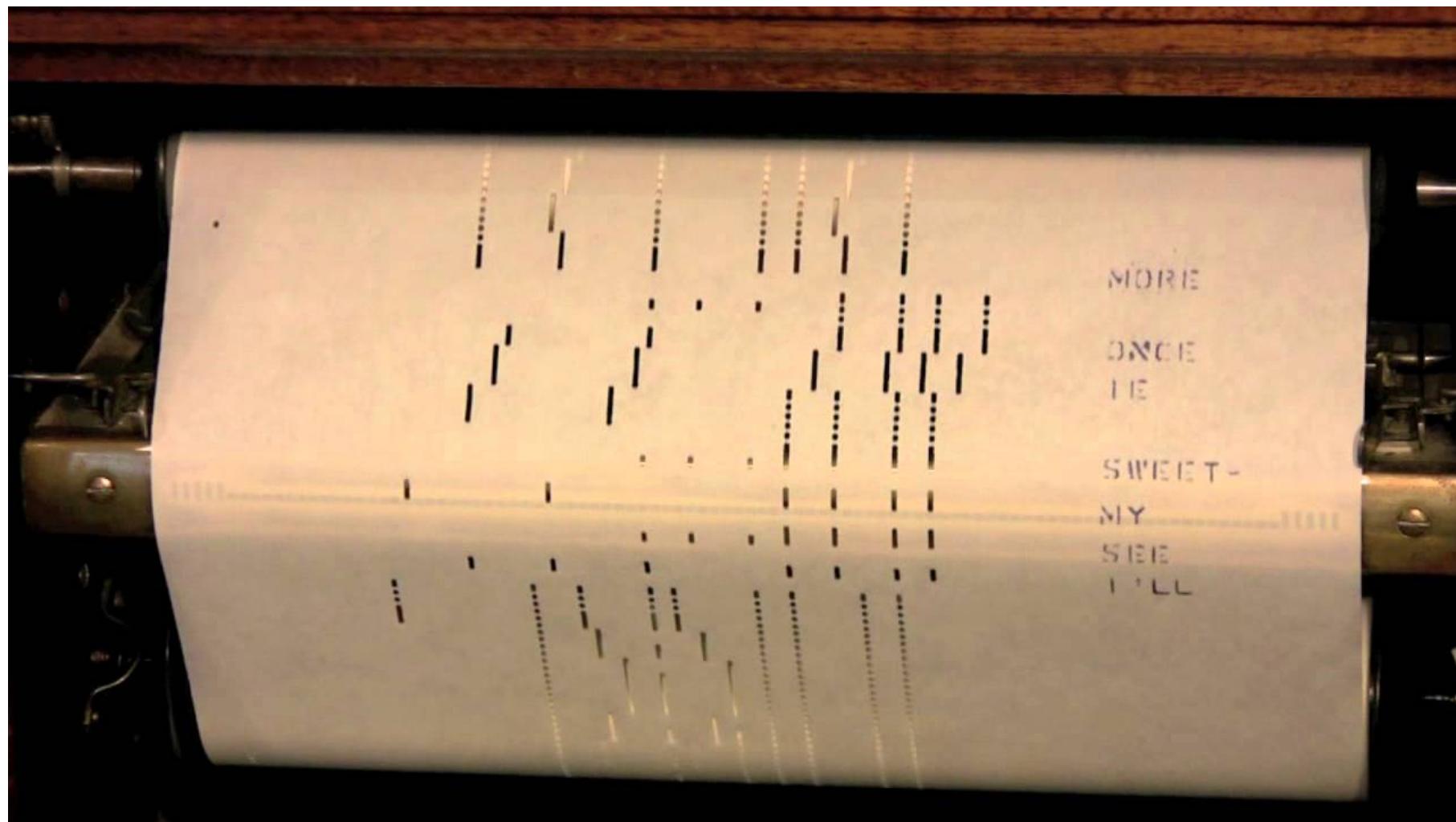
Punch cards (and legacy systems)

- How we used to enter programs into the computer



- Pre-computer origins: 18-th century use in textile looms
 - Responsible for the 80-character legacy

Who knows what this is?



The point: Information is more than data

Data

- raw, unorganized facts that need to be processed
- unusable until it is organized

Information

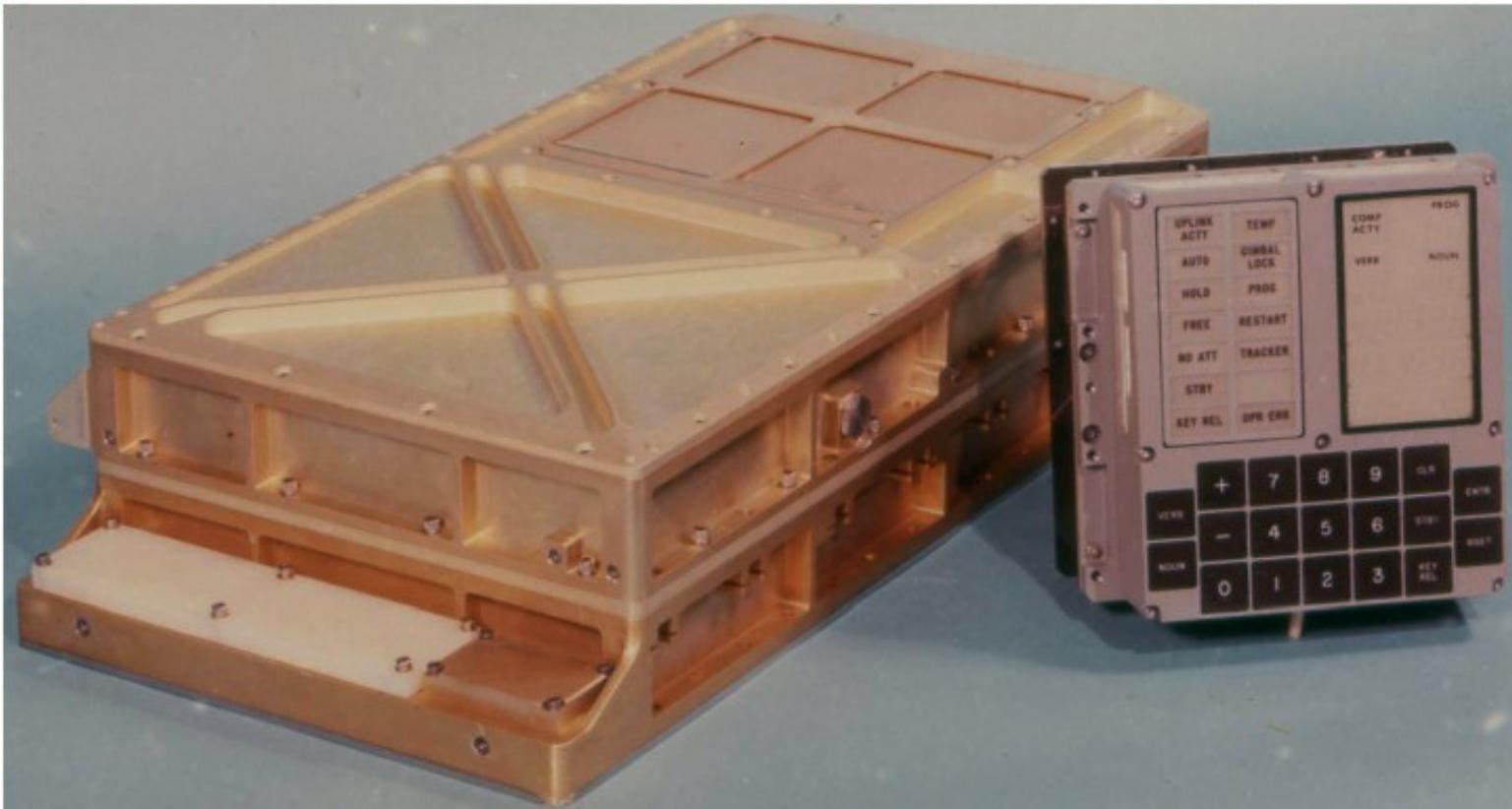
- created when data is processed, organized, structured
- needs to be situated in an appropriate *context* in order to become useful

There are important differences in how humans and computers treat data as information

Changes in the world of data

- volume of data in the modern world: 90% of the world's data **generated in the last two years**
- and that was *in 2013*
- SKA: Square Kilometer Array
 - a southern hemisphere radio telescope with a total of 1 sq km of data sensors
 - will generate 1 exabyte *daily* =
$$1 \times 10^{18}$$
 bytes
 - = 1,000,000,000,000,000,000 bytes

- compare this with the Apollo Guidance Computer (1966), which guided the first humans to the moon
 - 16-bit wordlength, 2048 words RAM (magnetic core memory) = *4KB*
 - 36,864 words ROM (core rope memory) = *73KB*



Basic units of data

- Bits
 - smallest unit of storage, a 0 or 1
 - anything that can store two states - now "transistors", used to be vacuum tubes
 - with n bits, can store 2^n patterns - so one byte can store 256 patterns

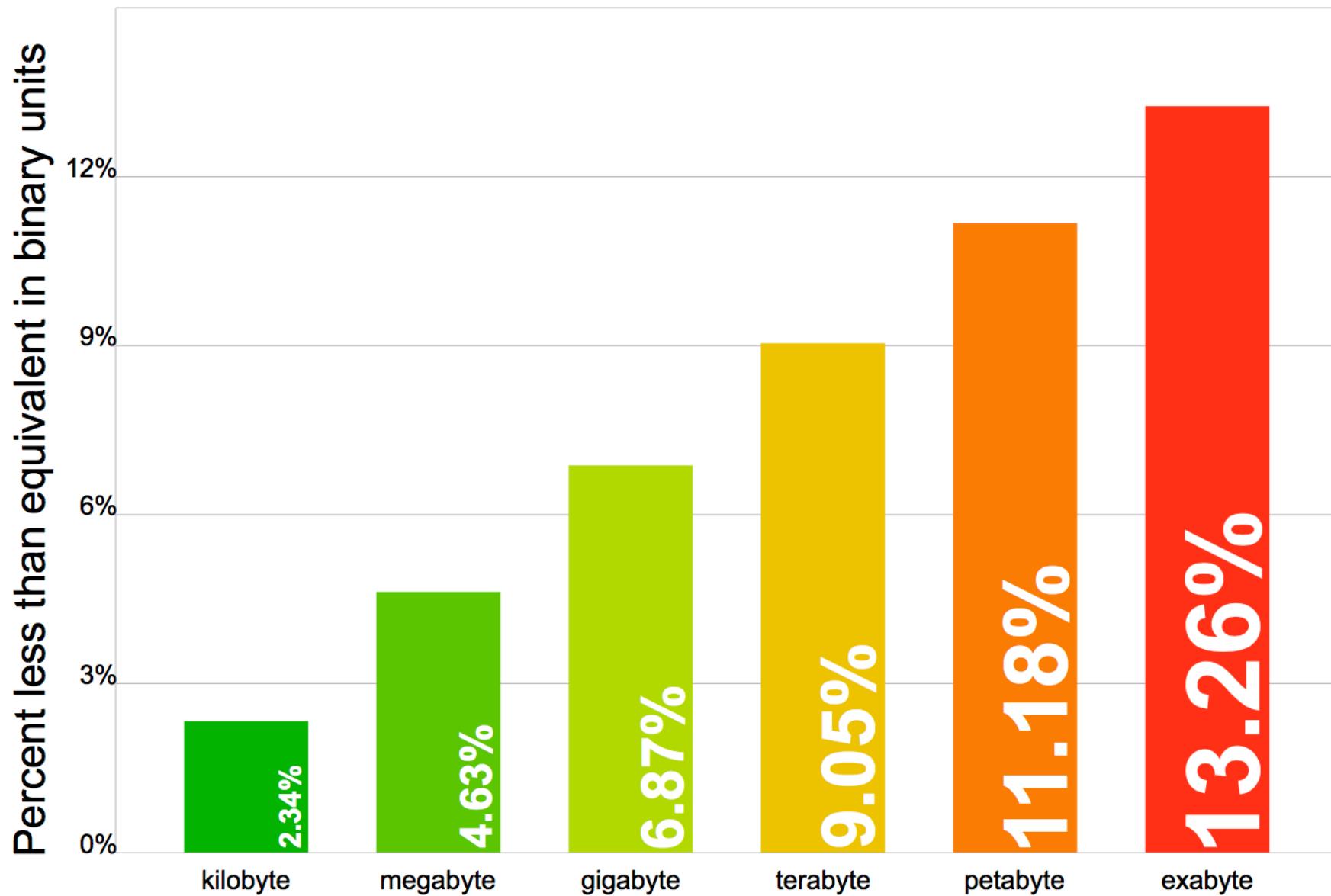
- Bytes
 - *eight bits = one byte*
 - "eight bit encoding" - represented characters, such as A represented as 65

multi-byte units

unit	abbreviation	total bytes	nearest decimal equivalent
kilobyte	KB	1,024^1	1000^1
megabyte	MB	1,024^2	1000^2
gigabyte	GB	1,024^3	1000^3
terabyte	TB	1,024^4	1000^4
petabyte	PB	1,024^5	1000^5
exabyte	EB	1,024^6	1000^6
zettabyte	ZB	1,024^7	1000^7
yottabyte	YB	1,024^8	1000^8

- this is why 1GB is greater than 1 billion bytes

Comparison of decimal and binary units



Data types: Generically

- objects are *bound* to an identifier, e.g.

```
temperature = 98.6
print(temperature)
```

98.6

- here, temperature is a variable name assigned to the literal floating-point object with the value of 98.6
- in Python, this is an instance of the **float** class
- identifiers in R and Python are *case-sensitive*
- some identifiers are typically reserved, e.g.

```
False, True, None, or, and    # Python
FALSE, TRUE, NA               # R
```

- All programming languages use comments, for humans to read
 - this is anything that follows the # character in both Python and R

"Let us change our traditional attitude to the construction of programs:
Instead of imagining that our main task is to instruct a computer what to do,
let us concentrate rather on explaining to human beings what we want a
computer to do." -- Donald Knuth, *Literate Programming* (1984)

Instantiation

- objects have *classes*, meaning they represent a "type" of object
- *instantiation* means creating a new instance of that class
- "immutable" objects cannot be subsequently changed

Python class	Immutable	Description	R class
bool	Yes	Boolean value	logical
int	Yes	integer number	integer
float	Yes	floating-point number	numeric
list	No	mutable sequence of objects	list
tuple	Yes	immutable sequence of objects	-
str	Yes	character string	character
set	No	unordered set of distinct objects	-
frozenset	Yes	immutable form of set class	-
dict	No	dictionary	(named) list

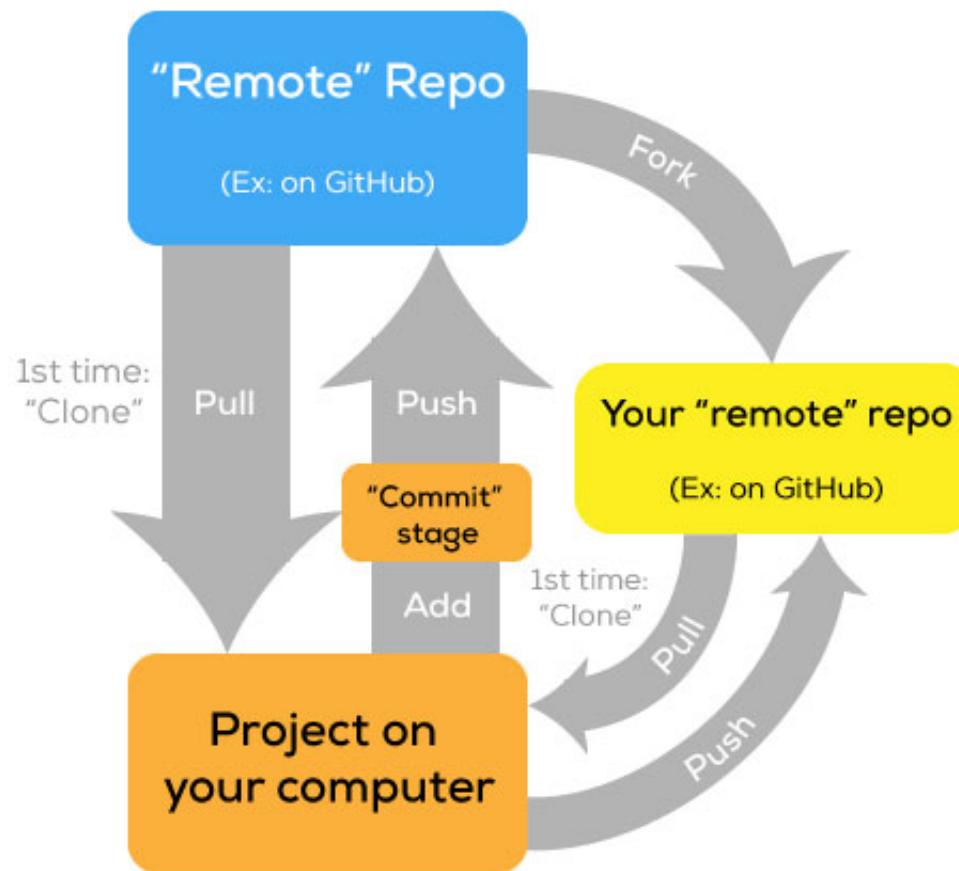
git

- git: a revision control system
- Allows for complete history of changes, branching, staging areas, and flexible and distributed workflows

- simplified workflow (from Anita Cheng's excellent blog post)

Git for non-developers

(in a tiny nutshell)



- Works through the command line, or through GUI clients, or through most IDEs and (good) editors

- or, RStudio

- or through editors (here, the excellent Atom editor)

The screenshot shows the Atom editor with a Git integration interface. The main window displays the file `index.md` with a diff view. The diff shows changes made to the file, with some lines highlighted in red (deletions) and green (additions). The right side of the interface contains a sidebar for managing staged changes, with a section titled "Staged Changes" showing the file `index.md` has been staged. Below the diff, there is a text input field labeled "Update handout". At the bottom, there are buttons for "Amend" and "Commit", along with other git-related controls like "LF", "master", and a file count indicator "1 file".

GitHub

- a website and hosting platform for git repositories
- and so much more...
 - publishing websites: <http://kenbenoit.net>, whose source code is at <https://github.com/kbenoit/kbenoit.github.io>
 - "continuous integration" hooks: <https://github.com/kbenoit/spacyr> (for instance - see the badges)
 - Issue tracking: <https://github.com/kbenoit/spacyr/issues>
 - Inspecting code: (e.g.) <https://github.com/kbenoit/spacyr/blob/master/R/python-functions.R>
- GitHub classroom
- Free stuff for students! <https://education.github.com/pack>

More great resources for using git/GitHub

- An easy git Cheatsheet, by Nina Jaeschke and Roger Dudler
- git - the simple guide by Roger Dudler

Some people have entire, open-source, user-commented books online, such as:

- *R for Data Science*
 - with source code [here](#)
 - with GitHub issues [here](#)
 - and pull requests - [examples here](#)

Markdown (and other markup languages)

- Idea of a "markup" language: HTML, XML, LaTeX
- "Markdown"
 - Created by John Gruber as a simple way for non-programming types to write in an easy-to-read format that could be converted directly into HTML
 - No opening or closing tags
 - Plain text, and can be read when not rendered
- Markdown has many "flavours"

Markdown example

This is a markdown example.

- bullet list 1
- bullet list 2

"I love deadlines. I like the whooshing sound they make as they fly by."
-- Douglas Adams

```
# Markdown example
```

```
This is a markdown example
```

```
bullet list 1  
bullet list 2
```

```
> "[I love deadlines. I like the whooshing sound they make as they fly by.](https://ww  
-- _Douglas Adams_
```

A good reference for Markdown: <https://ia.net/writer/support/general/markdown-guide/>.

A note on "meta-data"

- Data that provides information about other (primary) data
- Usually not meant to be analyzed as data itself
- Example: HTML

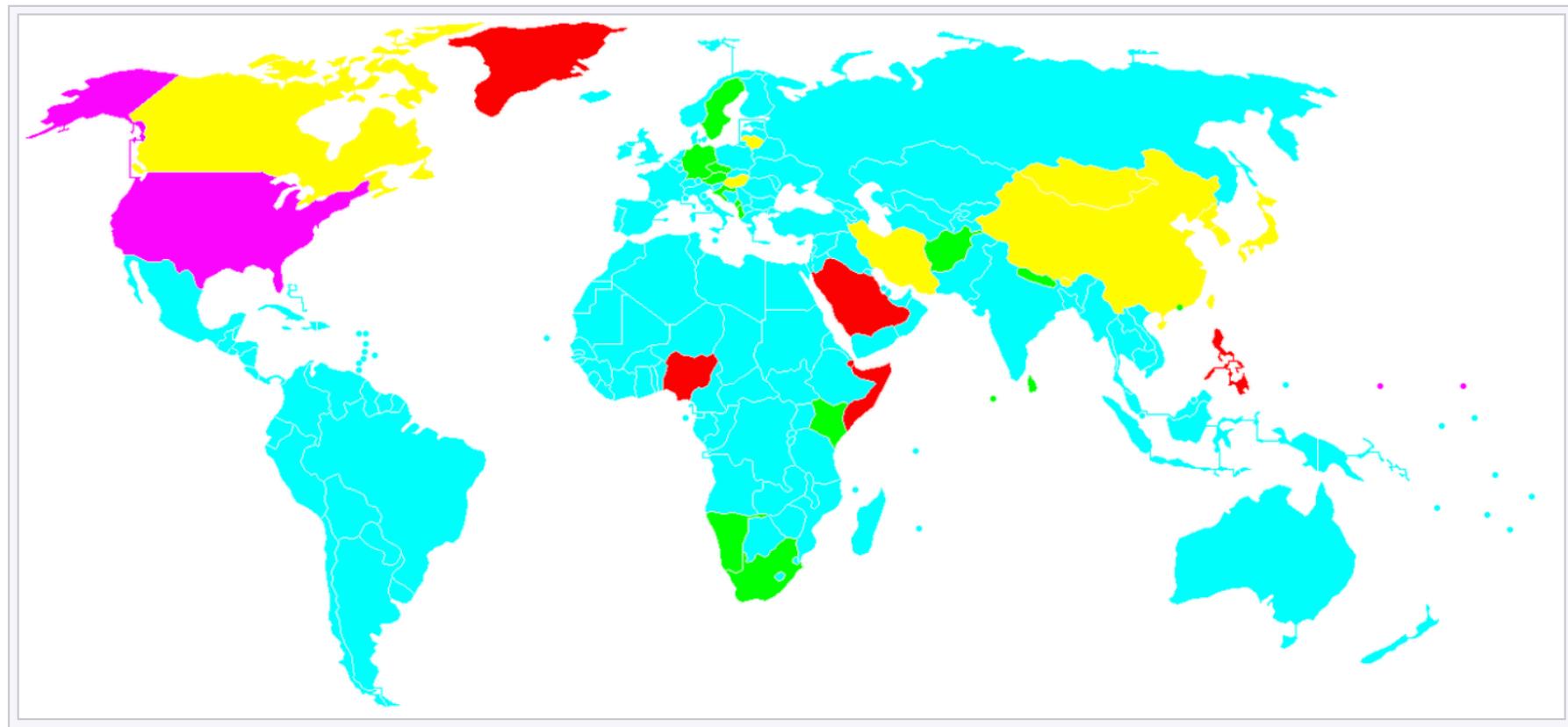
```
<!DOCTYPE html>
<html class="client-nojs" lang="en" dir="ltr">
<head>
<meta charset="UTF-8"/>
<title>Metadata – Wikipedia</title>
```

- Example of a standard attempting to address this need: [Dublin Core Metadata Initiative](#)

Representing dates: Different formats?

Color	Order styles	Main regions and countries (approximate population of each region in millions)	Approximate population in millions
Cyan	DMY	Asia (Central, SE, West), Australia (25), New Zealand (5), parts of Europe (c. 640), Latin America (625), North Africa (195), India (1315), Indonesia (265), Bangladesh (165), Russia (145)	3565
Yellow	YMD	Bhutan, Canada (35), China (1385), Koreas (75), Taiwan (24), Hungary (10), Iran (80), Japan (125), Lithuania (5), Mongolia (5). Known in other countries due to ISO 8601.	1745
Magenta	MDY	United States (325), Federated States of Micronesia, Marshall Islands	325
Red	DMY, MDY	Malaysia (35), Nigeria (190), Philippines (105), Saudi Arabia (35), Somalia (10)	380
Green	DMY, YMD	Afghanistan (28), Albania (3), Austria (9), Czech Republic (11), Kenya (49), Macau (1), Maldives, Montenegro, Namibia (2), Nepal (29), Singapore (6), South Africa (56), Sri Lanka (21), Sweden (10) ^[1]	225

Representing dates: Different formats?



Representing dates

Description	Format	Examples
American month and day	mm/dd	"5/12", "10/27"
American month, day and year	mm/dd/y	"1/17/2006"
Four digit year, month and day with slashes	YY/mm/dd	"2008/6/30", "1978/12/22"
Four digit year and month (GNU)	YY-mm	"2008-6", "2008-06", "1978-12"
Year, month and day with dashes	y-mm-dd	"2008-6-30", "78-12-22", "8-6-21"
Day, month and four digit year	dd-mm-YY	"30-6-2008"
Day, month and two digit year	dd.mm.yy	"30.6.08"
Day, textual month and year	dd-m y	"30-June 2008"
Textual month and four digit year	m YY	"June 2008", "March 1879"
Four digit year and textual month	YY m	"2008 June"
Textual month, day and year	m dd, y	"April 17, 1790"

ISO8601: Imposing common standards

- Purpose: to provide unambiguous and well-defined method of representing dates and times
- Goal: to avoid misinterpretation of numeric representations of dates and times, particularly when data are transferred between countries with different conventions for writing numeric dates and time
- First published in 1988
- Introduces a common notation, and a common order (most-to-least-significant order [YYYY]-[MM]-[DD])
 - matches lexicographical order with chronological order
- Uses codes for date and time elements, to represent dates (and times) in either a basic format (no separators) or in an extended format with added separators (to enhance human readability)

ISO8601 formatting components

Symbol	Meaning	Example	Notes
YYYY	4-digit year	2017	Avoids the "Y2K" problem
MM	2-digit day of the month	10	
DD	2-digit day of the month	03	
Www	Week number	52	
D	Weekday number	2	Starts on Monday!
hh	hour (0-24)	10	24 is only used to denote midnight at the end of a calendar day
mm	minute (0-59)	05	
ss	second (0-60)	20	60 is only used to denote an added leap second

Coordinated Universal Time (UTC)

- World standard for time
- Does not include Daylight Savings Time
- Interchangeable with Greenwich Mean Time (GMT), but GMT is no longer precisely defined by the scientific community
- **Time zones around the world** are expressed using positive or negative offsets from UTC
- French v. English
 - English speakers originally proposed *CUT* (for "coordinated universal time")
 - French speakers proposed *TUC* (for "temps universel coordonné")
 - Compromise: *UTC*

POSIX Time

- a system for describing a point in time, defined as the number of seconds that have elapsed since 00:00:00 UTC, Thursday, 1 January 1970
- Also known as "Unix time", or "**epoch time**", because it represents elapsed time from a defined "epoch"
- Problem: How much elapsed time can you store?

The "Year 2038 Problem"

- Many Unix-like operating systems which keep time as seconds elapsed from the epoch date of January 1, 1970
- For signed 32-bit integers, this means that cannot encode times after 03:14:07 UTC on 19 January 2038
- Times beyond that will wrap around and be stored internally as a negative number, which these systems will interpret as having occurred on 13 December 1901
- A solution: 64-bit signed integers allow a new wraparound date that is 20x greater than the estimated age of the universe: approximately 290 billion years in the future

Dataset manipulation

What is a “Dataset”?

- A dataset *is* a “rectangular” formatted table of data in which all the values of the same variable must be in a single column
- A dataset is *not*:
 - the results of tabulating a dataset
 - any set of summary statistics on a dataset
 - a series of relational tables
- Many of the datasets we use have been artificially reshaped in order to fulfill this criterion of rectangularity
 - This means “non-normalized” data
 - Often confounds variables with their values

The difference between a dataset and table

- This is a *table*:

		Lost	Won
Challenger		266	60
Incumbent		32	106

- This is a (partial) dataset:

		district	incumbf	wonseatf
1	Carlow	Kilkenny	Challenger	Lost
2	Carlow	Kilkenny	Challenger	Lost
5	Carlow	Kilkenny	Incumbent	Won
100	Donegal	South West	Challenger	Lost
459		Wicklow	Incumbent	Won
464		Wicklow	Challenger	Lost

Hadley Wickham's three rules for "tidy" datasets

1. Each variable must have its own column.
2. Each observation must have its own row.
3. Each value must have its own cell.

country	year	cases	population
Afghanistan	1990	745	1987071
Afghanistan	2000	2666	2059360
Brazil	1999	37737	172006362
Brazil	2000	80488	174504898
China	1999	212258	1272915272
China	2000	216766	128042583

variables

country	year	cases	population
Afghanistan	1990	745	1987071
Afghanistan	2000	2666	2059360
Brazil	1999	37737	172006362
Brazil	2000	80488	174504898
China	1999	212258	1272915272
China	2000	216766	128042583

observations

country	year	cases	population
Afghanistan	1990	745	1987071
Afghanistan	2000	2666	2059360
Brazil	1999	37737	172006362
Brazil	2000	80488	174504898
China	1999	212258	1272915272
China	2000	216766	128042583

values

Example from *R for Data Science*

- Example of non-tidy data

```
suppressPackageStartupMessages(library("tidyverse"))
print(table4a)
```

```
## # A tibble: 3 x 3
##   country    `1999` `2000`
## * <chr>      <int>  <int>
## 1 Afghanistan    745    2666
## 2 Brazil        37737   80488
## 3 China         212258  213766
```

What we need to do: make this into "long" format

country	year	cases	country	1999	2000
Afghanistan	1999	745	Afghanistan	745	2666
Afghanistan	2000	2666	Brazil	37737	80488
Brazil	1999	37737	China	212258	213766
Brazil	2000	80488			
China	1999	212258			
China	2000	213766			

table4

Current recommended way: pivot_longer()

```
pivot_longer(table4a, c(`1999`, `2000`),  
            names_to = "year", values_to = "cases")  
  
## # A tibble: 6 x 3  
##   country     year   cases  
##   <chr>       <chr>   <int>  
## 1 Afghanistan 1999     745  
## 2 Afghanistan 2000    2666  
## 3 Brazil      1999  37737  
## 4 Brazil      2000  80488  
## 5 China       1999 212258  
## 6 China       2000 213766
```

Previously: "gathering"

```
gather(table4a, `1999`, `2000`, key = "year", value = "cases")
```

```
## # A tibble: 6 x 3
##   country     year   cases
##   <chr>       <chr>   <int>
## 1 Afghanistan 1999     745
## 2 Brazil       1999   37737
## 3 China        1999  212258
## 4 Afghanistan 2000    2666
## 5 Brazil       2000  80488
## 6 China        2000 213766
```

Reshaping to wide format: pivot_wider()

```
print(table2)

## # A tibble: 12 x 4
##   country     year type      count
##   <chr>       <int> <chr>     <int>
## 1 Afghanistan 1999 cases      745
## 2 Afghanistan 1999 population 19987071
## 3 Afghanistan 2000 cases     2666
## 4 Afghanistan 2000 population 20595360
## 5 Brazil      1999 cases     37737
## 6 Brazil      1999 population 172006362
## 7 Brazil      2000 cases     80488
## 8 Brazil      2000 population 174504898
## 9 China       1999 cases     212258
## 10 China      1999 population 1272915272
## 11 China      2000 cases     213766
## 12 China      2000 population 1280428583
```

Objective: make long column into wide columns

country	year	key	value	country	year	cases	population
Afghanistan	1999	cases	745	Afghanistan	1999	745	19987071
Afghanistan	1999	population	19987071		2000	2666	20595360
Afghanistan	2000	cases	2666	Brazil	1999	37737	172006362
Afghanistan	2000	population	20595360		2000	80488	174504898
Brazil	1999	cases	37737	China	1999	212258	1272915272
Brazil	1999	population	172006362		2000	213766	1280428583
Brazil	2000	cases	80488				
Brazil	2000	population	174504898				
China	1999	cases	212258				
China	1999	population	1272915272				
China	2000	cases	213766				
China	2000	population	1280428583				

table2

```
pivot_wider(table2, names_from = type, values_from = count)

## # A tibble: 6 x 4
##   country     year   cases population
##   <chr>      <int>   <int>       <int>
## 1 Afghanistan 1999     745 19987071
## 2 Afghanistan 2000    2666 20595360
## 3 Brazil       1999  37737 172006362
## 4 Brazil       2000  80488 174504898
## 5 China        1999 212258 1272915272
## 6 China        2000 213766 1280428583
```

Older method: spread()

```
spread(table2, key = type, value = count)
```

```
## # A tibble: 6 x 4
##   country     year   cases population
##   <chr>      <int>  <int>      <int>
## 1 Afghanistan 1999    745  19987071
## 2 Afghanistan 2000   2666  20595360
## 3 Brazil       1999  37737 172006362
## 4 Brazil       2000  80488 174504898
## 5 China        1999 212258 1272915272
## 6 China        2000 213766 1280428583
```

But you should use the current `pivot_*`() functions in **tidyR**.

Example with Manifesto Data

```
load("cmpdata.Rdata")
glimpse(cmpdata)

## # Rows: 709
## # Columns: 83
## # $ country      <dbl> 42, 42, 42, 42, 42, 42, 42, 42, 42, 42, 42, 42, ...
## # $ countryname   <chr> "Austria", "Austria", "Austria", "Austria", "...
## # $ oecdmember    <int> 10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 10, ...
## # $ eumember       <int> 0, 0, 0, 0, 0, 0, 0, 10, 10, 10, 10, 10, 10, 1...
## # $ edate         <date> 1990-10-07, 1990-10-07, 1990-10-07, 1990-10-07, 1994-10...
## # $ date          <int> 199010, 199010, 199010, 199010, 199410, 199410, ...
## # $ party          <dbl> 42420, 42110, 42320, 42520, 42420, 42421, 42110, 42320, ...
## # $ partyname     <chr> "FPÖ Freedom Party", "GA Green Alternative", "SPÖ Social...
## # $ parfam         <int> 40, 10, 30, 50, 40, 40, 10, 30, 50, 50, 10, 30, 40, 40, ...
## # $ coderid        <int> 202, 202, 202, 202, 203, 203, 203, 203, 203, 203, 203, 2...
## # $ coderyear      <int> 1991, 1991, 1991, 1991, 1998, 1998, 1998, 1998, 1998, 19...
## # $ manual         <int> 1, 1, 1, 1, NA, NA, NA, NA, NA, NA, NA, NA, 999, ...
## # $ testresult     <dbl> 0.8160, 0.8160, 0.8160, 0.8160, 0.7650, 0.7650, 0.7650, ...
## # $ pervote        <dbl> 16.634764, 4.784040, 42.780708, 32.064484, 22.497438, 5...
## # $ voteest        <int> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...
## # $ presvote       <dbl> 998, 998, 998, 998, 998, 998, 998, 998, 998, 998, 998, 998, 9...
## # $ absseat        <int> 33, 10, 80, 60, 42, 11, 13, 65, 52, 53, 9, 71, 40, 10, 5...
```

select just ID variables and the codes

```
cmpdata <- cmpdata %>%  
  select(countryname, date, partyname, starts_with("per")) %>%  
  select(-pervote)  
  
glimpse(cmpdata)
```

```

## Rows: 709
## Columns: 60
## $ countryname <chr> "Austria", "Austria", "Austria", "Austria", "Austria", "Au...
## $ date       <int> 199010, 199010, 199010, 199010, 199410, 199410, 199410, 19...
## $ partyname  <chr> "FPÖ Freedom Party", "GA Green Alternative", "SPÖ Social D...
## $ per101     <dbl> 0, 0, 0, 5, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 7, 0...
## $ per102     <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1...
## $ per103     <dbl> 0, 2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 11, 0, ...
## $ per104     <dbl> 37, 0, 7, 6, 7, 0, 0, 13, 13, 0, 0, 0, 2, 0, 20, 0, 5, 40, ...
## $ per105     <dbl> 1, 3, 2, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 12, 6, 1, 2, ...
## $ per106     <dbl> 1, 0, 4, 3, 0, 0, 3, 3, 9, 0, 1, 5, 0, 0, 3, 2, 14, 8, 4, ...
## $ per107     <dbl> 16, 17, 44, 22, 3, 0, 2, 26, 62, 3, 1, 11, 0, 0, 10, 2, 11...
## $ per108     <dbl> 3, 0, 5, 8, 1, 0, 2, 6, 24, 1, 6, 11, 0, 0, 20, 0, 32, 149...
## $ per109     <dbl> 0, 0, 3, 0, 1, 0, 0, 0, 1, 0, 2, 4, 0, 0, 9, 0, 0, 6, 0, 0...
## $ per110     <dbl> 0, 3, 0, 0, 11, 0, 1, 0, 0, 0, 6, 0, 1, 0, 15, 1, 3, 7, 0, ...
## $ per201     <dbl> 12, 79, 22, 1, 109, 19, 20, 13, 36, 0, 12, 9, 11, 4, 101, ...52

```

```
cmpdata_long <- pivot_longer(cmpdata, cols = starts_with("per"),
                             names_to = "code", values_to = "value")
cmpdata_long

## # A tibble: 40,413 x 5
##   countryname    date partyname      code value
##   <chr>        <int> <chr>       <chr>  <dbl>
## 1 Austria      199010 FPÖ Freedom Party per101     0
## 2 Austria      199010 FPÖ Freedom Party per102     0
## 3 Austria      199010 FPÖ Freedom Party per103     0
## 4 Austria      199010 FPÖ Freedom Party per104     37
## 5 Austria      199010 FPÖ Freedom Party per105     1
## 6 Austria      199010 FPÖ Freedom Party per106     1
## 7 Austria      199010 FPÖ Freedom Party per107    16
## 8 Austria      199010 FPÖ Freedom Party per108     3
## 9 Austria      199010 FPÖ Freedom Party per109     0
## 10 Austria     199010 FPÖ Freedom Party per110     0
## # ... with 40,403 more rows
```

add a year variable

```
cmpdata_long <- mutate(cmpdata_long, year = floor(date / 100))  
head(cmpdata_long)
```

```
## # A tibble: 6 x 6  
##   countryname    date partyname      code  value  year  
##   <chr>        <int> <chr>       <chr>  <dbl>  <dbl>  
## 1 Austria      199010 FPÖ Freedom Party per101     0  1990  
## 2 Austria      199010 FPÖ Freedom Party per102     0  1990  
## 3 Austria      199010 FPÖ Freedom Party per103     0  1990  
## 4 Austria      199010 FPÖ Freedom Party per104    37  1990  
## 5 Austria      199010 FPÖ Freedom Party per105     1  1990  
## 6 Austria      199010 FPÖ Freedom Party per106     1  1990
```

Compare pro- versus anti- EU sentiment

- per108: European Community/Union: Positive
- per110: European Community/Union: Negative

See details at the [Manifesto Project website](#)

```
cmpdata_long2 <- cmpdata_long %>%
  group_by(code, year) %>%
  summarize(total = sum(value)) %>%
  filter(code %in% c("per108", "per110"))
```

`summarise()` has grouped output by 'code'. You can override using the `.groups` ar

```
cmpdata_long2
```

```
## # A tibble: 44 x 3
## # Groups:   code [2]
##   code     year total
##   <chr>   <dbl> <dbl>
## 1 per108  1990    170
## 2 per108  1991    650
## 3 per108  1992    232
## 4 per108  1993    921
## 5 per108  1994    659
## 6 per108  1995    498
## 7 per108  1996   1964
## 8 per108  1997    311
## 9 per108  1998    492
## 10 per108 1999   1470
## # ... with 34 more rows
```

```
ggplot(cmpdata_long2, aes(x = year, y = total,  
                           colour = code, group = code)) +  
  geom_line() +  
  scale_colour_manual(name = "EU Sentiment",  
                      labels = c("Positive", "Negative"),  
                      values =c("darkgreen", "red"))
```