The Shape of Data

ME314: Introduction to Data Science and Machine Learning

Ken Benoit

12 July 2022

Plan today

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

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

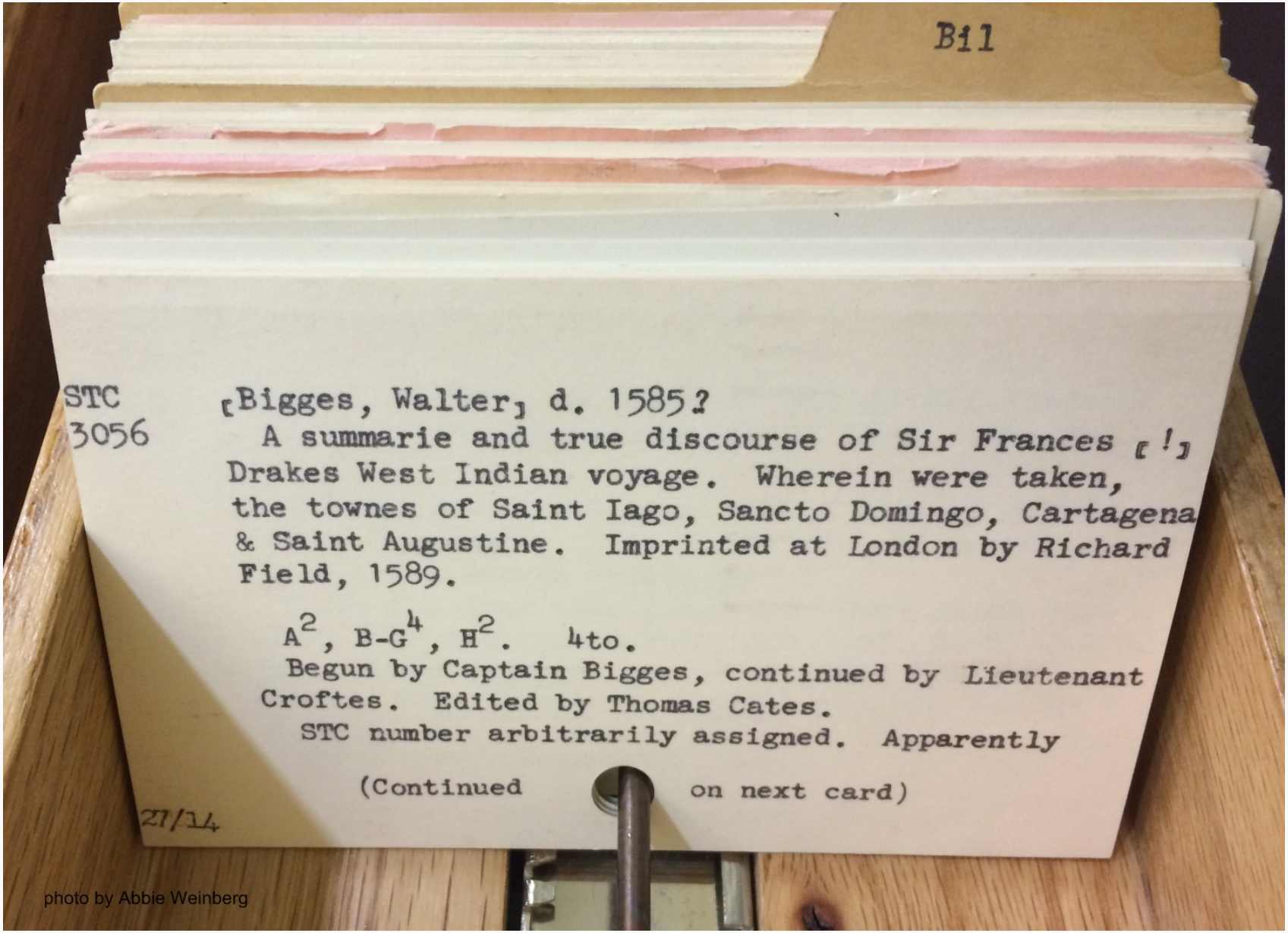
Structured data: Index cards

- 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

Heyday: Use in libraries to catalog books

a record looked like this



Dewey decimal system

Dewey decimal system

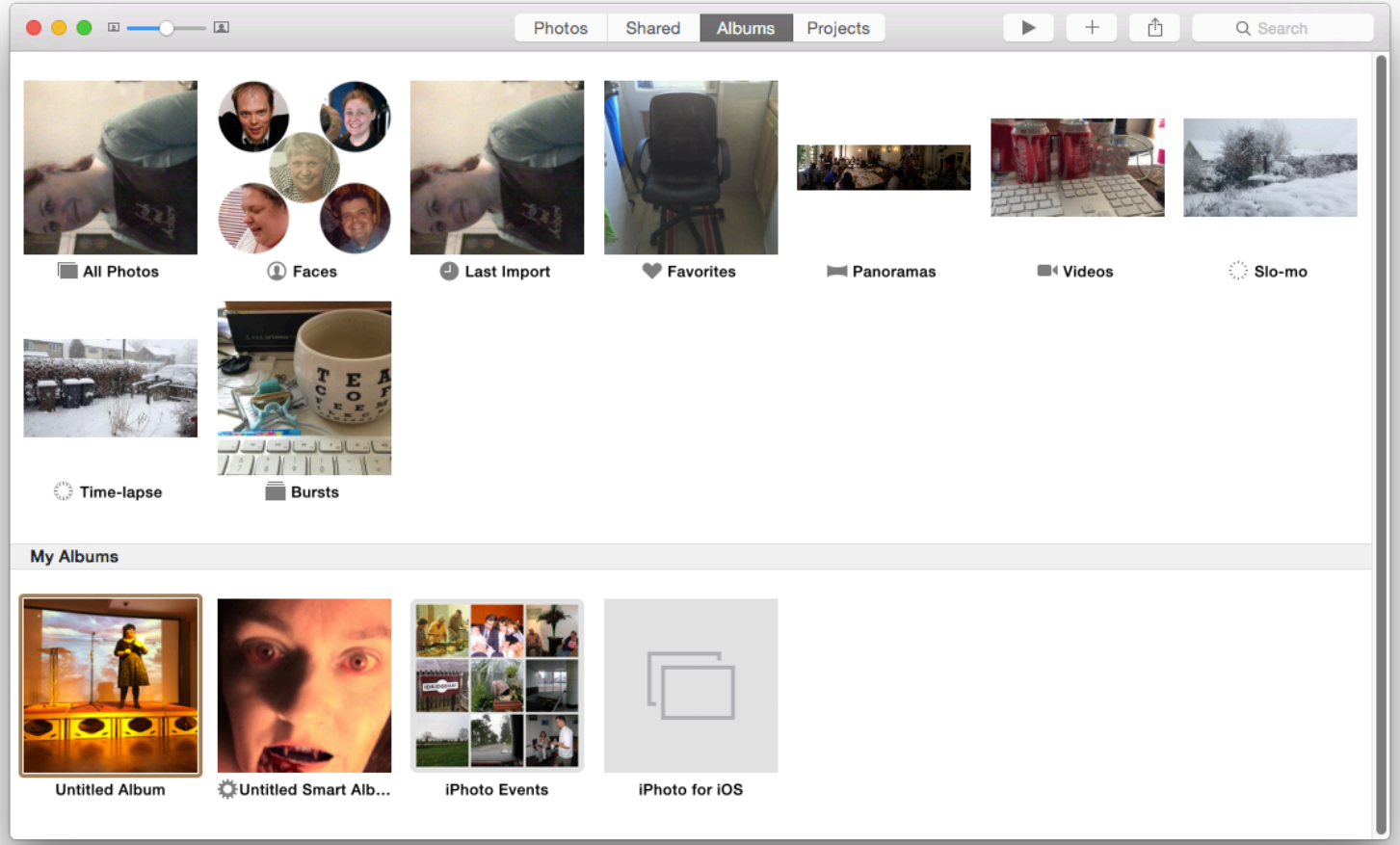
- a proprietary library classification system first published in the United States by Melvil Dewey in 1876
- scheme is made up of ten classes, each divided into ten divisions, each having ten sections
- the system's notation uses Arabic numbers, with three whole numbers making up the main classes and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

How to index?

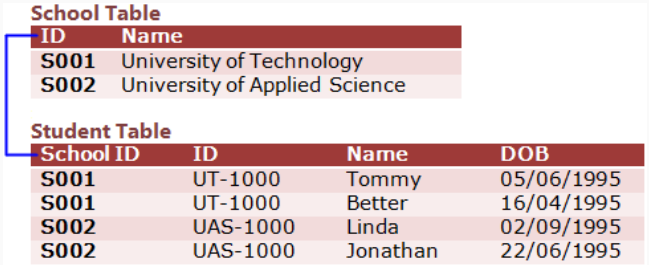
- Problem: Could only sort the cards one way
- Re-referencing was literally a manual operation
- Contrast with the idea of electronic indexes, where assets are stored once, and many indexing and reference systems can be applied



Modern database manager

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

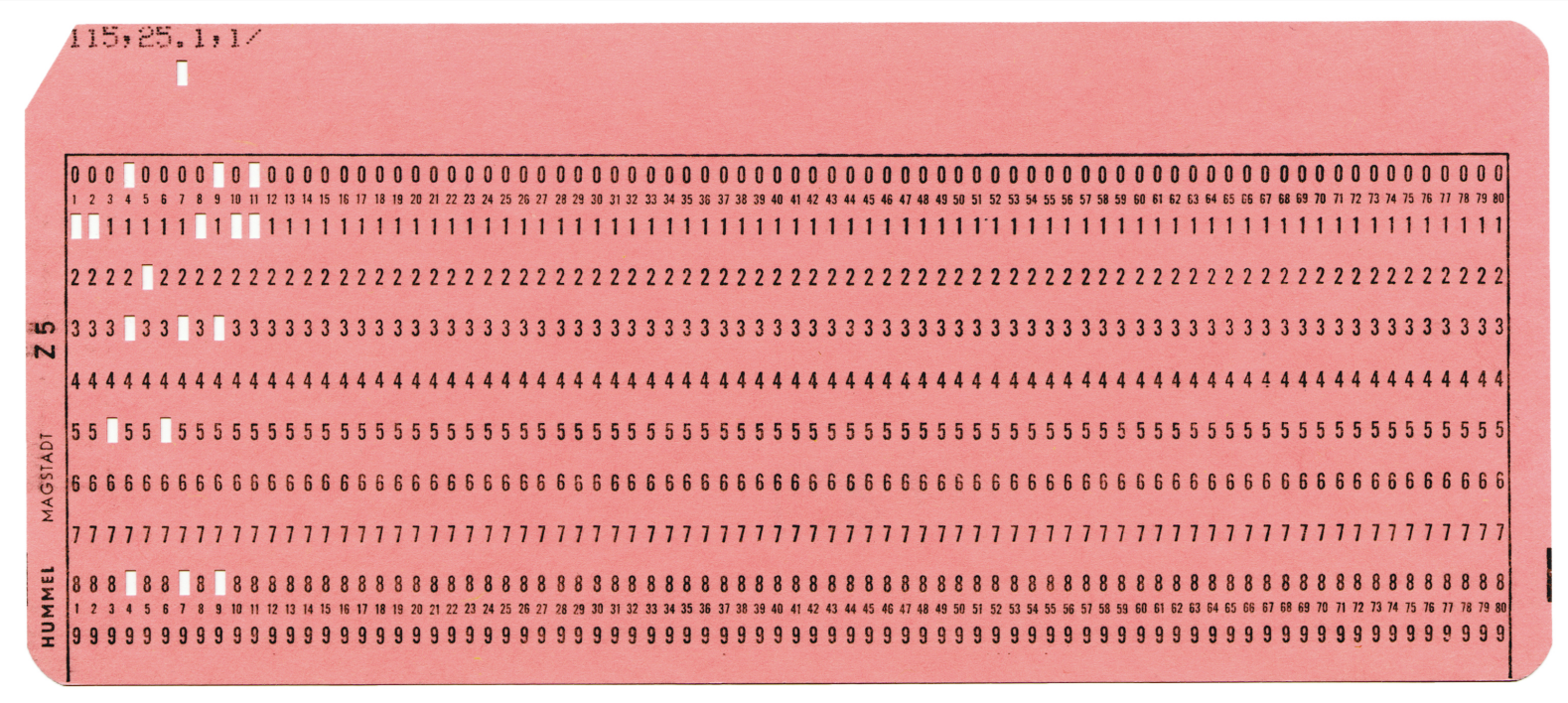
Student Table

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)

Punch cards (and legacy systems)

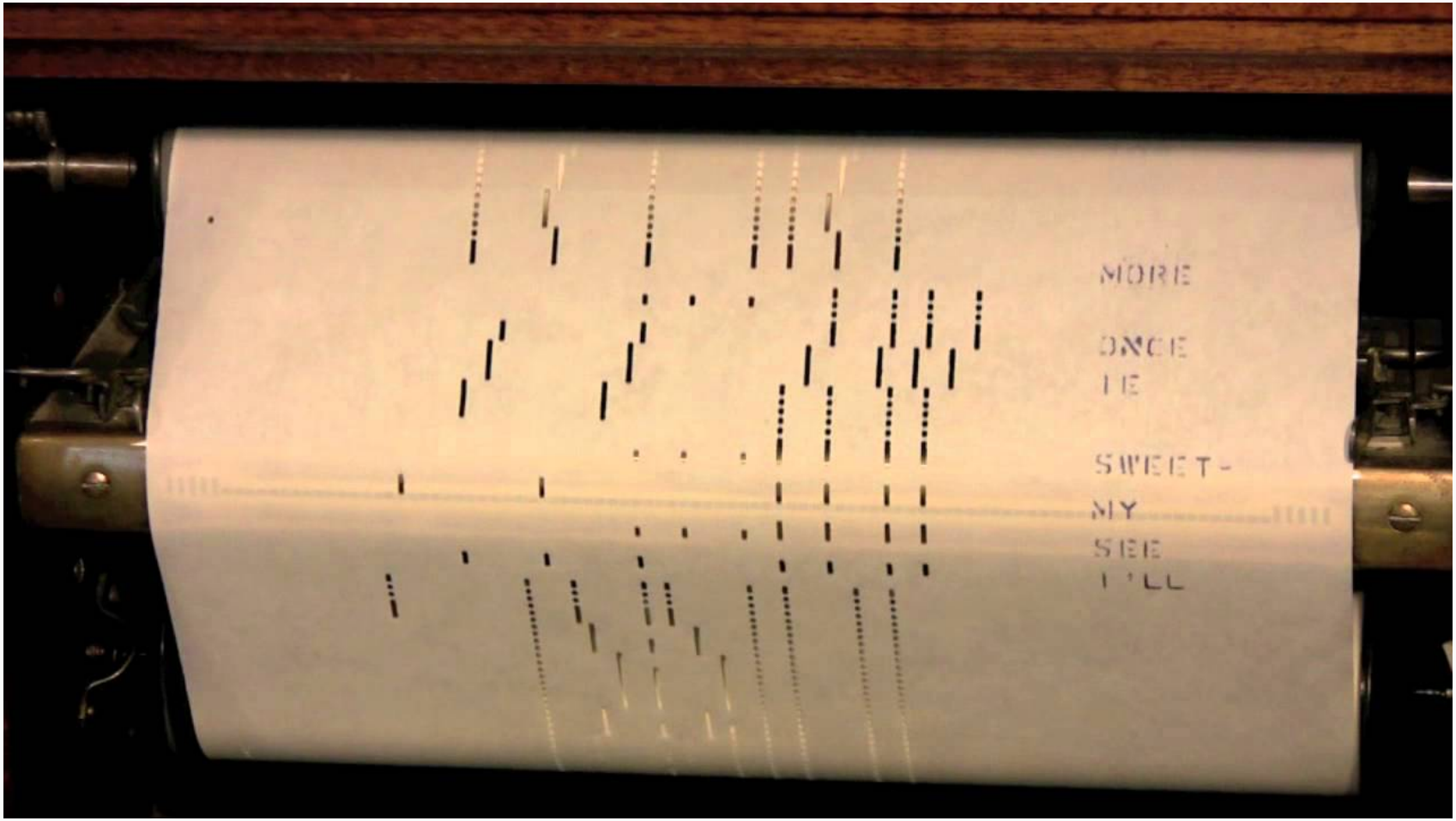
- How we used to enter programs into the computer



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

Who knows what this is?

Who knows what this is?



That is a piano roll, where music to be played by a player piano is encoded.

The point: Information is more than data

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

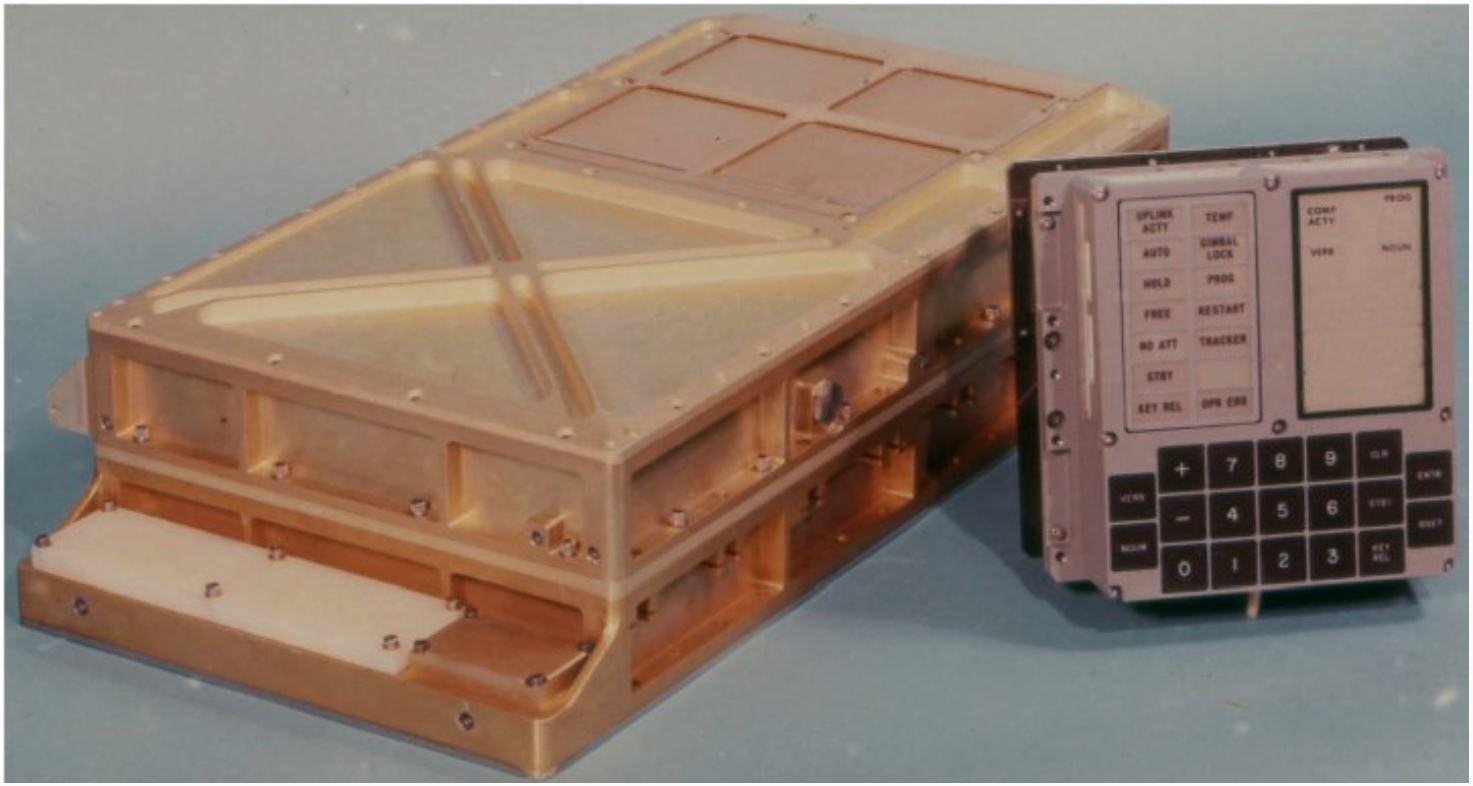
Changes in the world of data

- 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

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

Dec	Hx	Oct	Char	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr
0	0	000	NUL (null)	32	20	040		Space	64	40	100	@	@	96	60	140	`	`
1	1	001	SOH (start of heading)	33	21	041	!	!	65	41	101	A	A	97	61	141	a	a
2	2	002	STX (start of text)	34	22	042	"	"	66	42	102	B	B	98	62	142	b	b
3	3	003	ETX (end of text)	35	23	043	#	#	67	43	103	C	C	99	63	143	c	c
4	4	004	EOT (end of transmission)	36	24	044	$	$	68	44	104	D	D	100	64	144	d	d
5	5	005	ENQ (enquiry)	37	25	045	%	%	69	45	105	E	E	101	65	145	e	e
6	6	006	ACK (acknowledge)	38	26	046	&	&	70	46	106	F	F	102	66	146	f	f
7	7	007	BEL (bell)	39	27	047	'	'	71	47	107	G	G	103	67	147	g	g
8	8	010	BS (backspace)	40	28	050	(	(	72	48	110	H	H	104	68	150	h	h
9	9	011	TAB (horizontal tab)	41	29	051	)	)	73	49	111	I	I	105	69	151	i	i
10	A	012	LF (NL line feed, new line)	42	2A	052	*	*	74	4A	112	J	J	106	6A	152	j	j
11	B	013	VT (vertical tab)	43	2B	053	+	+	75	4B	113	K	K	107	6B	153	k	k
12	C	014	FF (NP form feed, new page)	44	2C	054	,	,	76	4C	114	L	L	108	6C	154	l	l
13	D	015	CR (carriage return)	45	2D	055	-	-	77	4D	115	M	M	109	6D	155	m	m
14	E	016	SO (shift out)	46	2E	056	.	.	78	4E	116	N	N	110	6E	156	n	n
15	F	017	SI (shift in)	47	2F	057	/	/	79	4F	117	O	O	111	6F	157	o	o
16	10	020	DLE (data link escape)	48	30	060	0	0	80	50	120	P	P	112	70	160	p	p
17	11	021	DC1 (device control 1)	49	31	061	1	1	81	51	121	Q	Q	113	71	161	q	q
18	12	022	DC2 (device control 2)	50	32	062	2	2	82	52	122	R	R	114	72	162	r	r
19	13	023	DC3 (device control 3)	51	33	063	3	3	83	53	123	S	S	115	73	163	s	s
20	14	024	DC4 (device control 4)	52	34	064	4	4	84	54	124	T	T	116	74	164	t	t
21	15	025	NAK (negative acknowledge)	53	35	065	5	5	85	55	125	U	U	117	75	165	u	u
22	16	026	SYN (synchronous idle)	54	36	066	6	6	86	56	126	V	V	118	76	166	v	v
23	17	027	ETB (end of trans. block)	55	37	067	7	7	87	57	127	W	W	119	77	167	w	w
24	18	030	CAN (cancel)	56	38	070	8	8	88	58	130	X	X	120	78	170	x	x
25	19	031	EM (end of medium)	57	39	071	9	9	89	59	131	Y	Y	121	79	171	y	y
26	1A	032	SUB (substitute)	58	3A	072	:	:	90	5A	132	Z	Z	122	7A	172	z	z
27	1B	033	ESC (escape)	59	3B	073	;	;	91	5B	133	[	[	123	7B	173	{	{
28	1C	034	FS (file separator)	60	3C	074	<	<	92	5C	134	\	\	124	7C	174			|
29	1D	035	GS (group separator)	61	3D	075	=	=	93	5D	135	]	]	125	7D	175	}	}
30	1E	036	RS (record separator)	62	3E	076	>	>	94	5E	136	^	^	126	7E	176	~	~
31	1F	037	US (unit separator)	63	3F	077	?	?	95	5F	137	_	_	127	7F	177	&#127;	DEL

Source: www.LookupTables.com

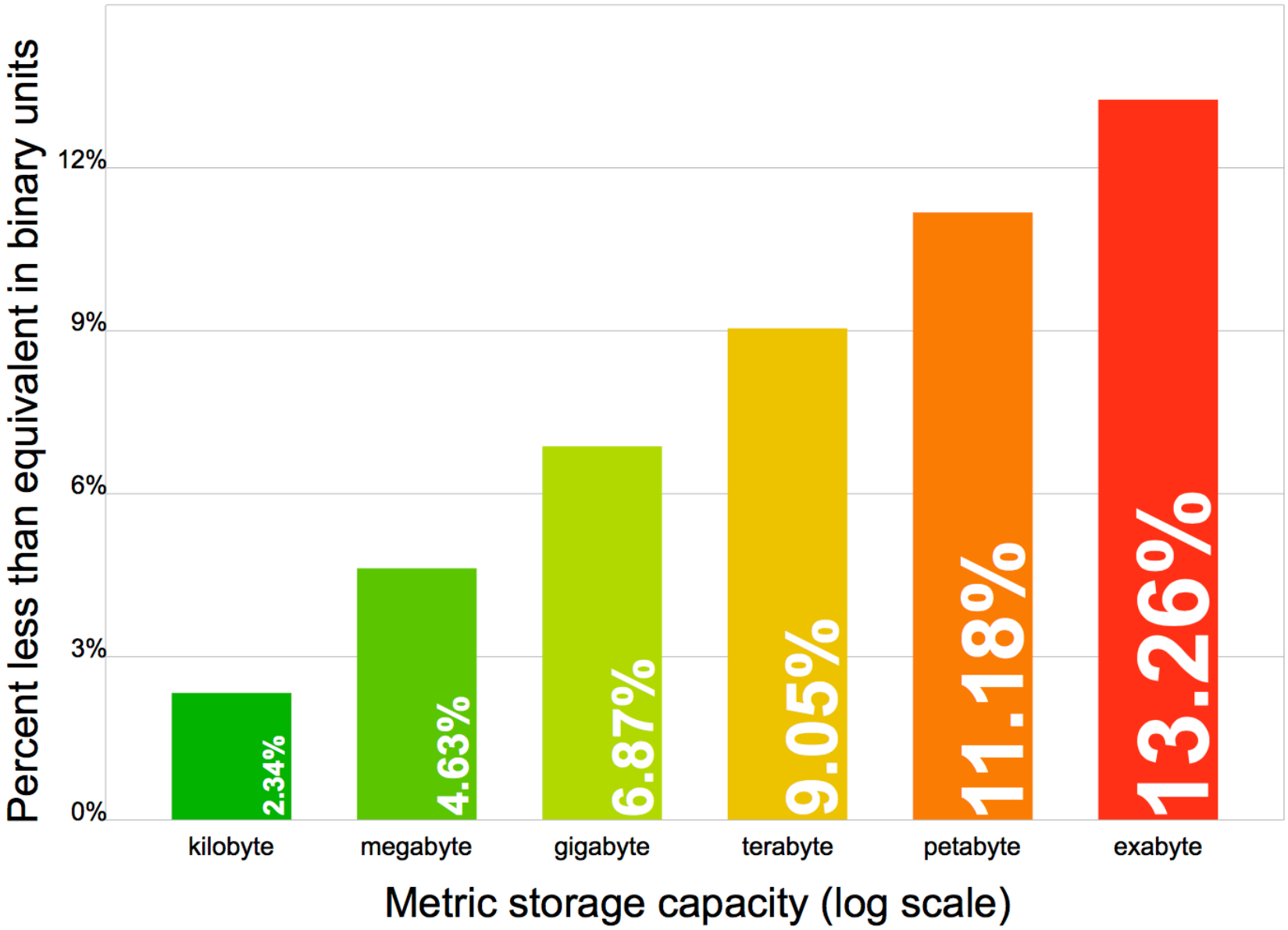
multi-byte units

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

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. `Python  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

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

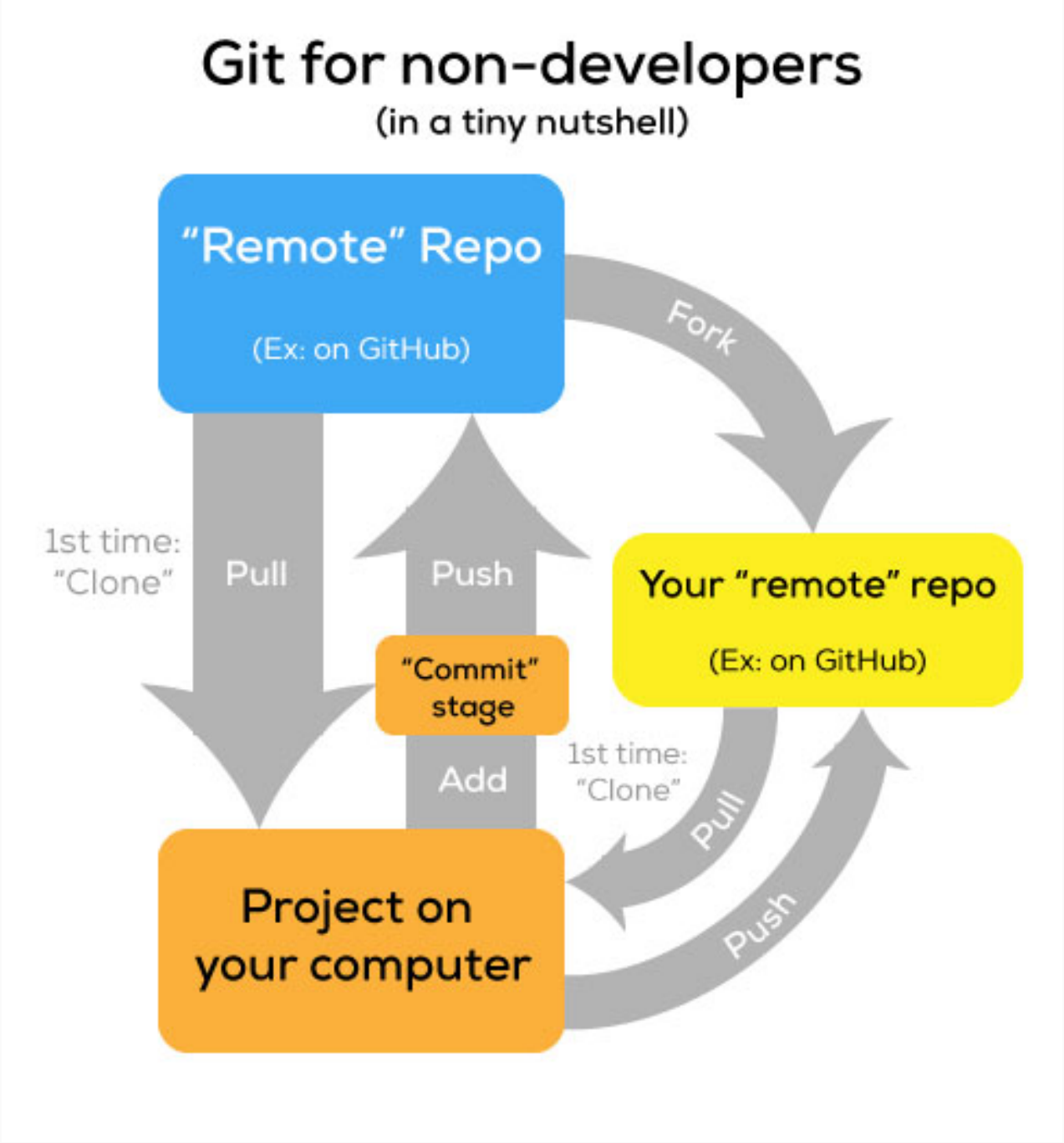
Python class	Immutable	Description	R class
bool	Yes	Boolean value	logical
int	Yes	integer number	integer
float	Yes	floating-point number	numeric
list	No	mutable sequence of objects	list
tuple	Yes	immutable sequence of objects	-
str	Yes	character string	character
set	No	unordered set of distinct objects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

git

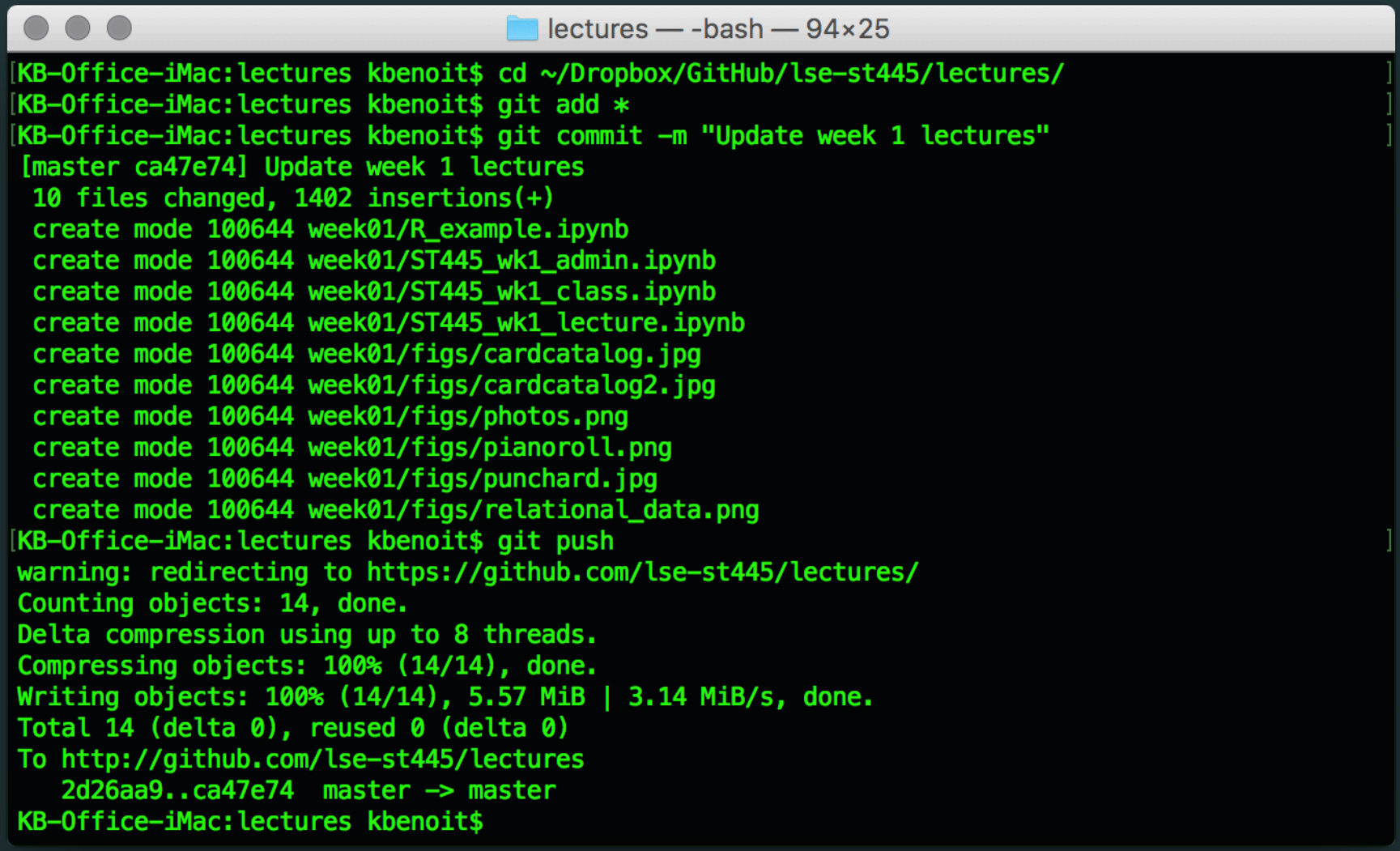
git

- `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)



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


lectures — -bash — 94×25
KB-Office-iMac:lectures kbenoit$ cd ~/Dropbox/GitHub/lse-st445/lectures/
KB-Office-iMac:lectures kbenoit$ git add *
KB-Office-iMac:lectures kbenoit$ git commit -m "Update week 1 lectures"
[master ca47e74] Update week 1 lectures
10 files changed, 1402 insertions(+)
create mode 100644 week01/R_example.ipynb
create mode 100644 week01/ST445_wk1_admin.ipynb
create mode 100644 week01/ST445_wk1_class.ipynb
create mode 100644 week01/ST445_wk1_lecture.ipynb
create mode 100644 week01/figs/cardcatalog.jpg
create mode 100644 week01/figs/cardcatalog2.jpg
create mode 100644 week01/figs/photos.png
create mode 100644 week01/figs/pianoroll.png
create mode 100644 week01/figs/punchard.jpg
create mode 100644 week01/figs/relational_data.png
KB-Office-iMac:lectures kbenoit$ git push
warning: redirecting to https://github.com/lse-st445/lectures/
Counting objects: 14, done.
Delta compression using up to 8 threads.
Compressing objects: 100% (14/14), done.
Writing objects: 100% (14/14), 5.57 MiB | 3.14 MiB/s, done.
Total 14 (delta 0), reused 0 (delta 0)
To http://github.com/lse-st445/lectures
2d26aa9..ca47e74 master -> master
KB-Office-iMac:lectures kbenoit$

- or, RStudio

GitHub

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
  - "continous 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

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)

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

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 th
-- _Douglas Adams_
```

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

A note on “meta-data”

A note on “meta-data”

- Data that provides infomation 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

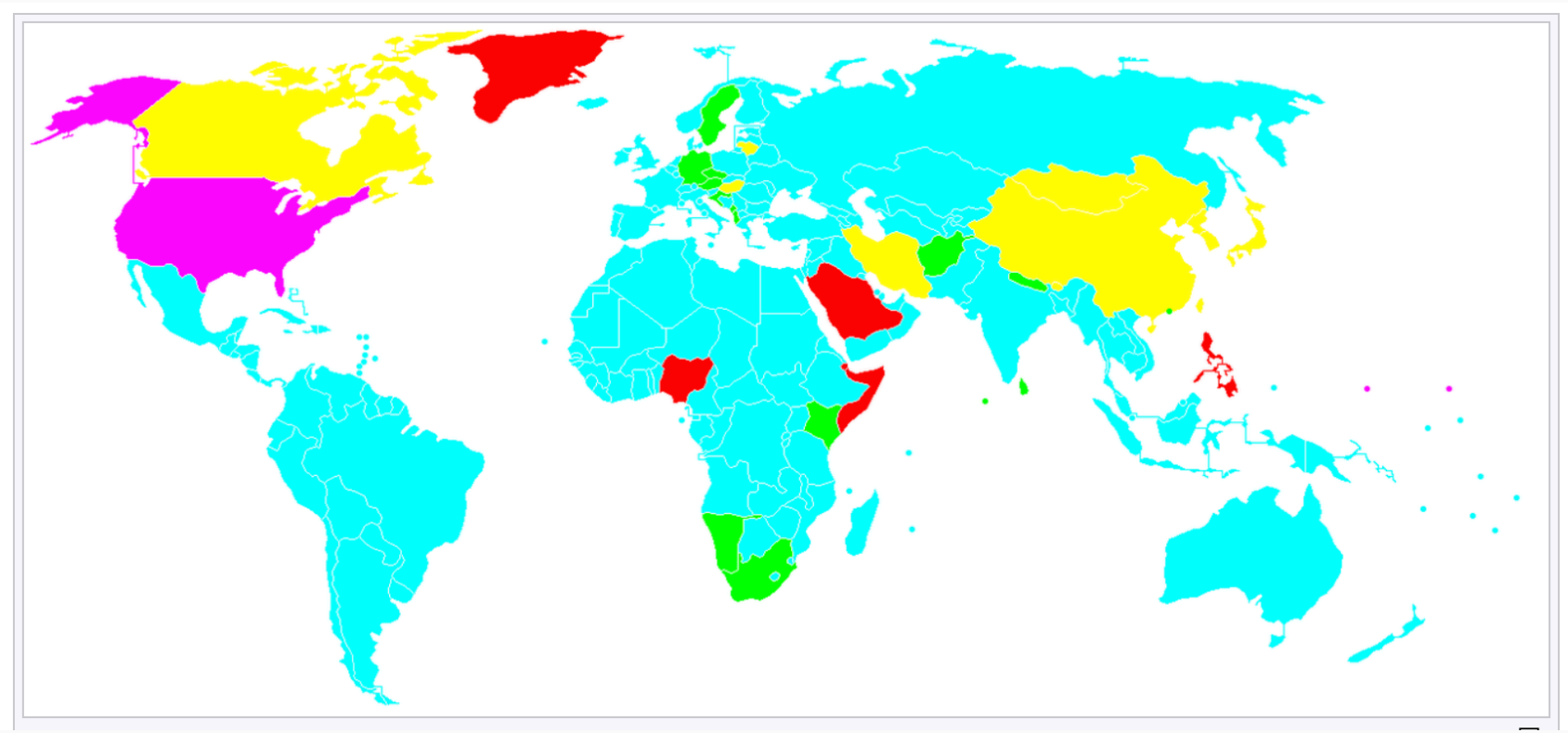
Representing dates: Different formats?

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: Different formats?



Representing dates

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	m YY	“June 2008”,

ISO8601: Imposing common standards

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

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

Coordinated Universal Time (UTC)

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

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 encoding 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

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

The difference between a dataset and table

- This is a *table*:

	Lost	Won
Challenger	266	60
Incumbent	32	106

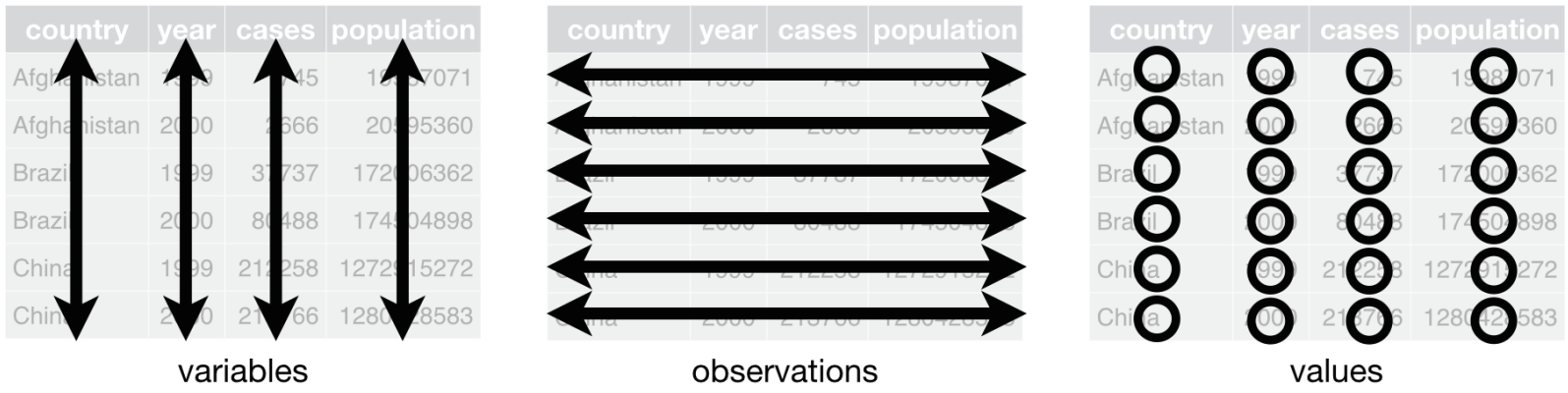
- This is a (partial) dataset:

	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

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.



Example from *R for Data Science*

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

Current recommended way: pivot_longer()

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”

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()

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
```

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

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,
## $ countryname   <chr> "Austria", "Austria", "Austria", "Austr
## $ oecdmember    <int> 10, 10, 10, 10, 10, 10, 10, 10, 10, 10,
## $ eumember      <int> 0, 0, 0, 0, 0, 0, 0, 0, 0, 10, 10, 10,
## $ edate         <date> 1990-10-07, 1990-10-07, 1990-10-07, 19
## $ date          <int> 199010, 199010, 199010, 199010, 199410,
## $ party         <dbl> 42420, 42110, 42320, 42520, 42420, 4242
## $ partyname     <chr> "FPÖ Freedom Party", "GA Green Alternat
## $ parfam        <int> 40, 10, 30, 50, 40, 40, 10, 30, 50, 50,
## $ coderid       <int> 202, 202, 202, 202, 203, 203, 203, 203,
## $ coderyear     <int> 1991, 1991, 1991, 1991, 1998, 1998, 199
## $ manual        <int> 1, 1, 1, 1, NA, NA, NA, NA, NA, NA, NA,
## $ testresult    <dbl> 0.8160, 0.8160, 0.8160, 0.8160, 0.7650,
```

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
## $ date        <int> 199010, 199010, 199010, 199010, 199410, 1
## $ partyname   <chr> "FPÖ Freedom Party", "GA Green Alternativ
## $ per101      <dbl> 0, 0, 0, 5, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
## $ per102      <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
## $ per103      <dbl> 0, 2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
## $ per104      <dbl> 37, 0, 7, 6, 7, 0, 0, 13, 13, 0, 0, 0, 2,
## $ per105      <dbl> 1, 3, 2, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0,
## $ per106      <dbl> 1, 0, 4, 3, 0, 0, 3, 3, 9, 0, 1, 5, 0, 0,
## $ per107      <dbl> 16, 17, 44, 22, 3, 0, 2, 26, 62, 3, 1, 11
```

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

Compare pro- versus anti- EU sentiment

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

See details at the Manifesto Project website

```r
cmpdata_long <- mutate(cmpdata_long, year = floor(date / 100))
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
## `.groups` argument.
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
```

```
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"))
```