

ST445 Managing and Visualizing Data

The Shape of Data

Week 2 Lecture, MT 2017 - Kenneth Benoit

Plan today

- Questions and administration
- Representing text data: Unicode
- Representing dates
- Representing sparse matrix formats
- Data and datasets
 - "tidy" data
 - reshaping data
 - normalization forms
- Lab preview

How to represent text data: encoding

- a “character set” is a list of character with associated numerical representations
- ASCII: the original character set, uses just 7 bits (2^7) – see http://ergoemacs.org/emacs/unicode_basics.html
(http://ergoemacs.org/emacs/unicode_basics.html)
- ASCII was later extended, e.g. ISO-8859
<http://www.ic.unicamp.br/~stolfi/EXPORT/www/ISO-8859-1-Encoding.html>
(<http://www.ic.unicamp.br/~stolfi/EXPORT/www/ISO-8859-1-Encoding.html>),
using 8 bits (2^8)
- but this became a jungle, with no standards:
http://en.wikipedia.org/wiki/Character_encoding
(http://en.wikipedia.org/wiki/Character_encoding)

Solution: Unicode

But: Unicode must still be *encoded*

- and displayed... Different devices will render displayed emoji differently (<https://unicode.org/emoji/charts/full-emoji-list.html>), for instance
- another problem: there are more far code points than fit into 8-bit encodings. Hence there are multiple ways to *encode* the Unicode code points
- *variable-byte* encodings use multiple bytes as needed. Advantage is efficiency, since most ASCII and simple extended character sets can use just one byte, and these were set in the Unicode standard to their ASCII and ISO-8859 equivalents
- two most common are UTF-8 and UTF-16, using 8 and 16 bits respectively

Doh! Byte Order Marks!

- Back to the big-endian v. little-endian issue: Where do you store the start of the data representation?

Example: UCS-2/UTF-16 (little-endian UCS-2 is the native format on Windows)

```
FF FE  4800 6500 6C00 6C00 6F00
header H    e    l    l    o
```

But in "big-endian":

```
FE FF  0048 0065 006C 006C 006F
header H    e    l    l    o
```

Doh! Byte Order Marks! (cont'd)

- Solution: A "byte-order mark" (U+FEFF) at the header of each file, indicating the byte order
 - FFFE for little-endian, FEFF for big-endian
- But this has a problem: The BOM is actually a valid Unicode character (<http://www.fileformat.info/info/unicode/char/FEFF/index.htm>)!
 - What if someone sent a file without a header, and that character was actually part of the file?

Solution: UTF-8

- One problem with fixed-length schemes (such as UTF-16) is that they are *wasteful*: they use two bytes even when this is unnecessary
- UTF-8 is designed to be more efficient: only as many bytes as you need

```
EF BB BF 48 65 6C 6C 6F
header   H  e  l  l  o
```

- Code points 0 – 007F are stored as regular, single-byte ASCII
- Code points 0080 and above are converted to binary and stored (encoded) in a series of bytes

Solution: UTF-8 (cont'd)

- UTF-8 is a *variable-length* encoding
 - The first “count” byte indicates the number of bytes for the codepoint, including the count byte. These bytes start with 11..0:

110xxxxx (The leading “11” indicates 2 bytes in sequence, including the “count” byte)
1110xxxx (1110 -> 3 bytes in sequence)
11110xxx (11110 -> 4 bytes in sequence)
 - Bytes starting with 10... are “data” bytes and contain information for the codepoint. A 2-byte example looks like this: 110xxxxx 10xxxxxx

This means there are 2 bytes in the sequence. The X's represent the binary value of the codepoint, which needs to squeeze in the remaining bits

- **Recommendation:** Use UTF-8 (<http://utf8everywhere.org>)

Text encoding: Caution

- Input texts can be very different
- Many text production software (e.g. MS Office-based products) still tend to use proprietary formats, such as Windows-1252
- Windows tends to use UTF-16, while Mac and other Unix-based platforms use UTF-8
- Your eyes can be deceiving: a client may display gibberish but the encoding might still be as intended
- No easy method of detecting encodings (except in HTML meta-data)

A note on "meta-data"

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

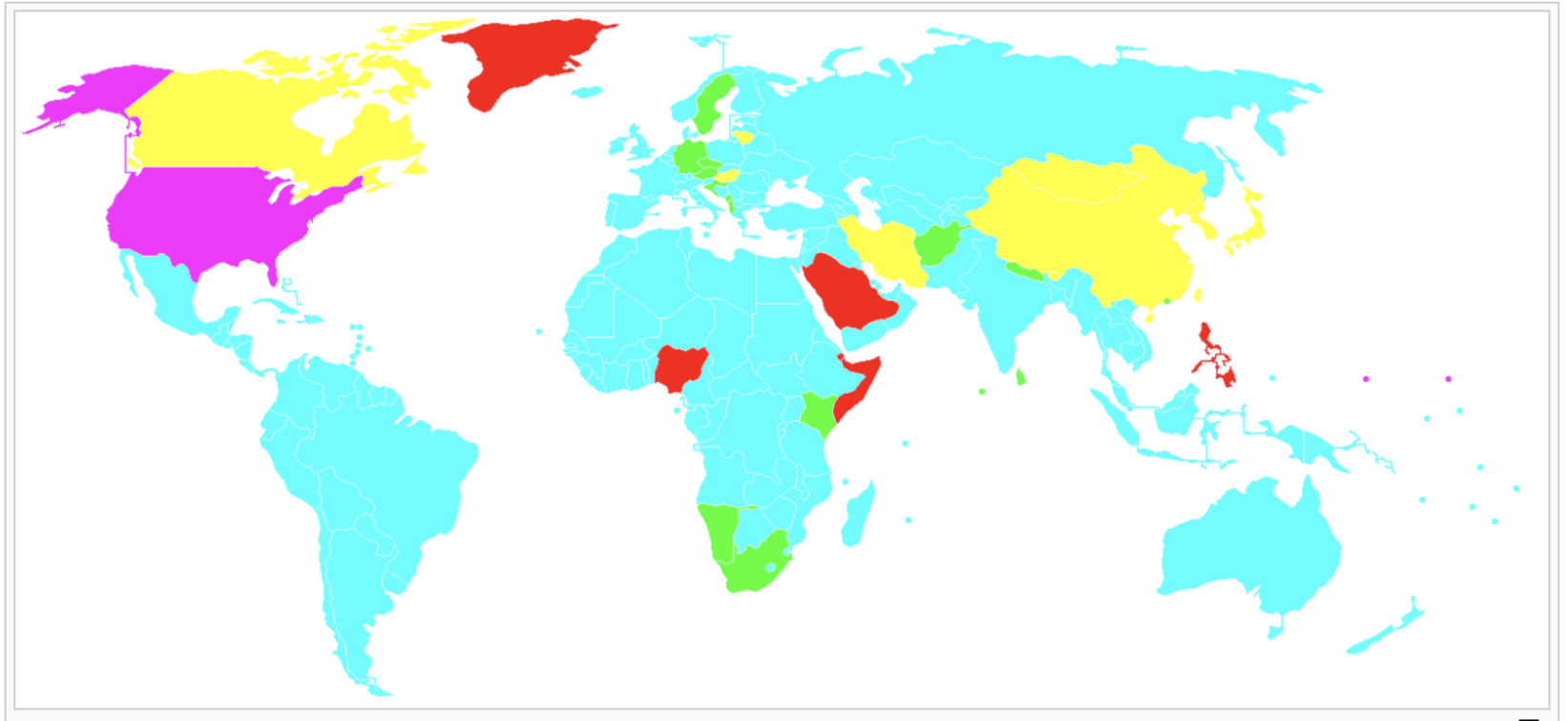
```
<!DOCTYPE html>  
<html class="client-nojs" lang="en" dir="ltr">  
<head>  
<meta charset="UTF-8"/>  
<title>Metadata - Wikipedia</title>
```

- Example of a standard attempting to address this need: Dublin Core Metadata Initiative (<http://dublincore.org/documents/dc-text/>)

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"
Day and textual month	d m	"1 July"
Month abbreviation, day and year	M-DD-y	"May-09-78", "Apr-17-1790"
Year, month abbreviation and day	y-M-DD	"78-Dec-22", "1814-MAY-17"

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
(https://en.wikipedia.org/wiki/List_of_UTC_time_offsets) 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"([https://en.wikipedia.org/wiki/Epoch_\(reference_date\)#Computing](https://en.wikipedia.org/wiki/Epoch_(reference_date)#Computing))), 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

Compression

- Seeks to economize on space by representing recurring items using patterns that represent the uncompressed data
- Common in formats for graphics and video encoding
- “Lossless” formats compress data without reducing information - examples are .zip and .gz compression
- This (and avoiding errors) is also a principle in normalized relational data forms
- Also very important for sparse matrix representations, where many of the cells are zero, but it would be very wasteful to record a double precision numeric zero for each of these non-informative cells

Compression: Sparse matrix formats

- used because many forms of matrix are very sparse - for example, document-term matrixes, which are commonly 80-90% sparse

```
In [1]: suppressPackageStartupMessages(library("quanteda"))  
mydfm <- dfm(data_corpus_inaugural)  
mydfm
```

Document-feature matrix of: 58 documents, 9,357 features (91.8% sparse).

```
In [22]: head(mydfm, nfeature = 8)
```

Document-feature matrix of: 6 documents, 8 features (37.5% sparse).
6 x 8 sparse Matrix of class "dfmSparse"

	features						
docs	fellow-citizens	of the	senate	and	house	representatives	:
1789-Washington	1	71	116	1	48	2	2 1
1793-Washington	0	11	13	0	2	0	0 1
1797-Adams	3	140	163	1	130	0	2 0
1801-Jefferson	2	104	130	0	81	0	0 1
1805-Jefferson	0	101	143	0	93	0	0 0
1809-Madison	1	69	104	0	43	0	0 0

"simple triplet" format

- "simple triplet" format
 - i : indexes row
 - j : indexes column
 - x : indicates value

(indexes will be from zero)

"simple triplet" example

This matrix:

	[,1]	[,2]	[,3]	[,4]
[1,]	1	99	3	12
[2,]	0	0	10	1
[3,]	2	0	0	0

Would be represented by:

- i : 0 2 0 0 1 0 1
- j : 0 0 1 2 2 3 3
- x : 1 2 99 3 10 12 1

In [13]: *# in R*

```
m <- matrix(c(1, 0, 2, 99, 0, 0, 3, 10, 0, 12, 1, 0), nrow = 3)
print(m)
```

```
      [,1] [,2] [,3] [,4]
[1,]     1    99     3    12
[2,]     0     0    10     1
[3,]     2     0     0     0
```

In [14]: **library**("Matrix")
m_st <- as(m, "dgTMatrix")
print(m_st@i)
print(m_st@j)
print(m_st@x)

```
[1] 0 2 0 0 1 0 1
[1] 0 0 1 2 2 3 3
[1] 1 2 99 3 10 12 1
```

"compressed sparse column" format

- More efficient than the STF
 - i : indexes row
 - p : indexes the first nonzero element in each column of the matrix
 - x : indicates value
- "compressed sparse row" format is also possible

"compressed sparse column" example

This matrix:

	[,1]	[,2]	[,3]	[,4]
[1,]	1	99	3	12
[2,]	0	0	10	1
[3,]	2	0	0	0

Would be represented by:

- i : 0 2 0 0 1 0 1
- p : 0 2 3 5 7
- x : 1 2 99 3 10 12 1

In [15]: *# in R*

```
m_sc <- as(m, "dgCMatrix")  
print(m_sc@i)  
print(m_sc@p)  
print(m_sc@x)
```

```
[1] 0 2 0 0 1 0 1
```

```
[1] 0 2 3 5 7
```

```
[1] 1 2 99 3 10 12 1
```

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 Kilkenney	Challenger	Lost
2	Carlow Kilkenney	Challenger	Lost
5	Carlow Kilkenney	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	1999	175	19987071
Afghanistan	2000	2666	20095360
Brazil	1999	37737	172006362
Brazil	2000	80488	174604898
China	1999	212258	1272915272
China	2000	216766	128042583

variables

country	year	cases	population
Afghanistan	1999	175	19987071
Afghanistan	2000	2666	20095360
Brazil	1999	37737	172006362
Brazil	2000	80488	174604898
China	1999	212258	1272915272
China	2000	216766	128042583

observations

country	year	cases	population
Afghanistan	99	75	19987071
Afghanistan	00	66	20095360
Brazil	99	77	172006362
Brazil	00	48	174604898
China	99	258	1272915272
China	00	76	128042583

values

Example from *R for Data Science*

- Example of non-tidy data

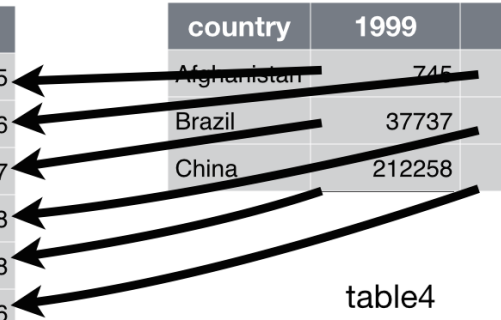
```
In [4]: suppressPackageStartupMessages(library("tidyverse"))  
        print(table4a)
```

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

"Gathering"

```
In [5]: print(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
```



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

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

table4

"Spreading"

```
In [6]: 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
```



```
In [7]: spread(table2, key = type, value = count) %>% print()
```

```
# 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
```

country	year	key	value
Afghanistan	1999	cases	745
Afghanistan	1999	population	19987071
Afghanistan	2000	cases	2666
Afghanistan	2000	population	20595360
Brazil	1999	cases	37737
Brazil	1999	population	172006362
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

Long v. wide formats (R)

- More fine-grained control with functions and packages for this (alternative to `dplyr` functions of `spread` `gather`, etc.)
- `base::reshape()`
 - the “old” R way to do this, using ‘`base::reshape()`’
 - problem: confusing and difficult to use
- `reshape2` package
 - Hadley Wickham’s `reshape2` package
 - data is first ‘melt’ed into long format
 - then ‘cast’ into desired format

Example with Manifesto Data (<https://manifesto-project.wzb.eu>)

```
In [37]: library("reshape2", warn.conflicts = FALSE)
load("cmpdata.Rdata")
head(cmpdata)
```

	country	countryname	oecdmember	eumember	edate	date	party
175	42	Austria	10	0	1990-10-07	199010	4242
176	42	Austria	10	0	1990-10-07	199010	4211
177	42	Austria	10	0	1990-10-07	199010	4232
178	42	Austria	10	0	1990-10-07	199010	4252
179	42	Austria	10	0	1994-10-09	199410	4242
180	42	Austria	10	0	1994-10-09	199410	4242

```
In [38]: cmpdataLong <- melt(cmpdata,
                             id.vars = c("countryname", "party", "date"),
                             measure.vars = names(cmpdata)[21:76],
                             variable.name = "category",
                             value.name = "catcount")
head(cmpdataLong, 30)
```

countryname	party	date	category	catcount
Austria	42420	199010	per101	0
Austria	42110	199010	per101	0
Austria	42320	199010	per101	0
Austria	42520	199010	per101	5
Austria	42420	199410	per101	0
Austria	42421	199410	per101	0
Austria	42110	199410	per101	0
Austria	42320	199410	per101	0
Austria	42520	199410	per101	0
Austria	42520	199512	per101	0
Austria	42110	199512	per101	0
Austria	42320	199512	per101	0
Austria	42420	199512	per101	0
Austria	42421	199512	per101	0

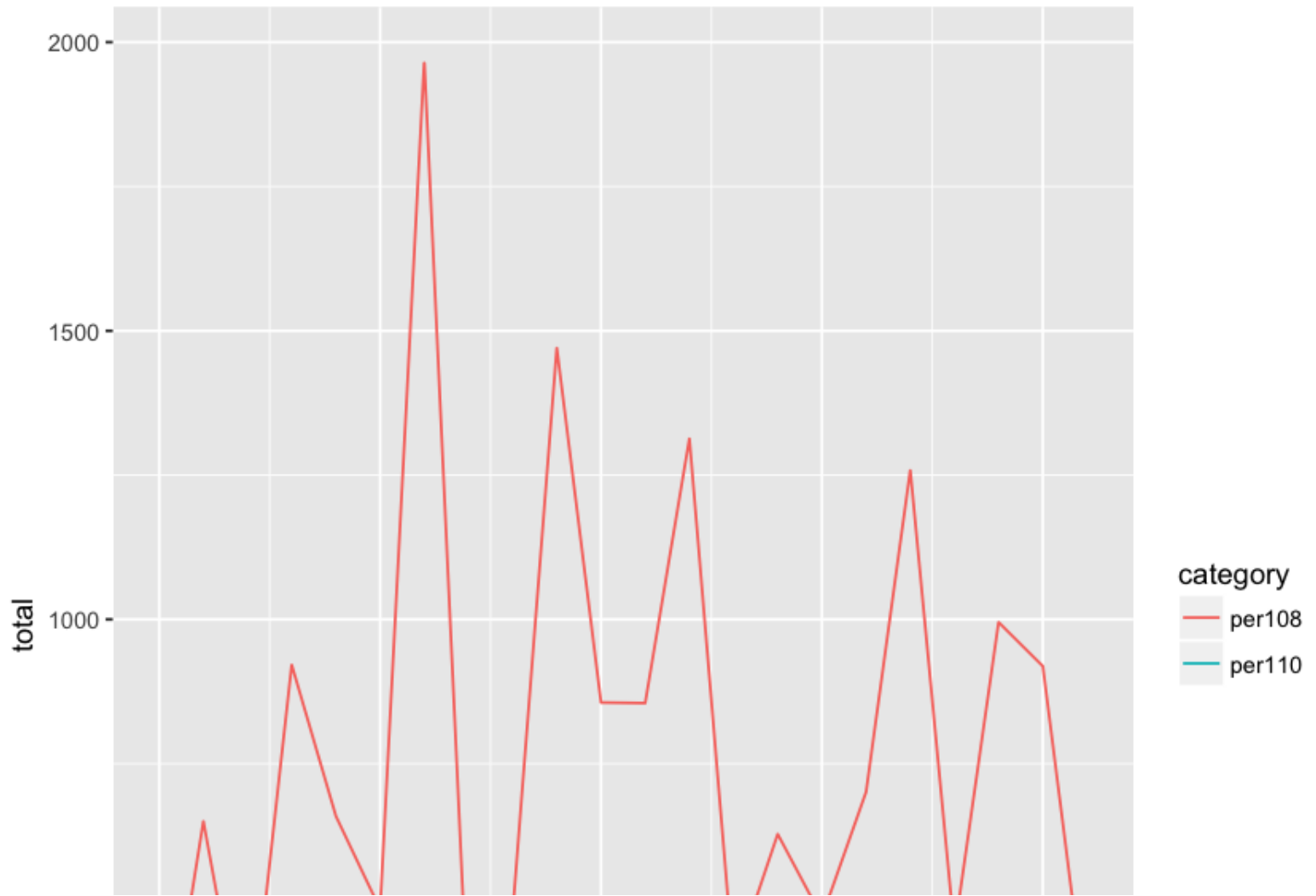
countryname	party	date	category	catcount
Austria	42420	199910	per101	1
Austria	42110	199910	per101	0
Austria	42320	199910	per101	0
Austria	42520	199910	per101	0
Austria	42520	200211	per101	7
Austria	42320	200211	per101	0
Austria	42110	200211	per101	0
Austria	42220	200211	per101	0
Austria	42420	200211	per101	0
Austria	42110	200610	per101	0
Austria	42710	200610	per101	0
Austria	42420	200610	per101	0
Austria	42520	200610	per101	0
Austria	42320	200610	per101	0
Austria	42320	200809	per101	0
Austria	42110	200809	per101	0

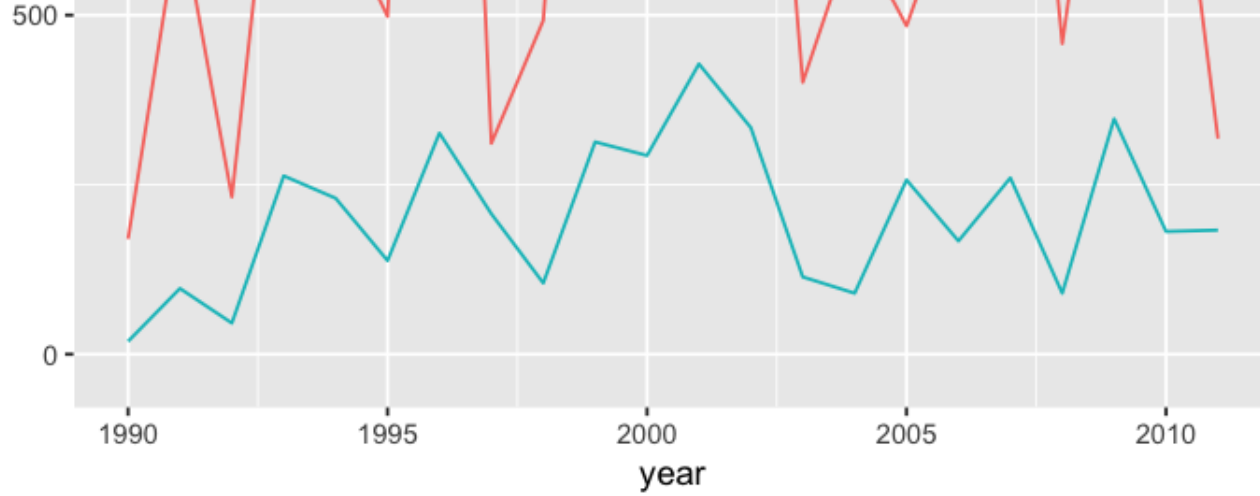
```
In [29]: tail(cmpdataLong, 30)
```

	countryname	party	date	category	catcount
39675	Switzerland	43530	199110	per706	0
39676	Switzerland	43420	199110	per706	4
39677	Switzerland	43810	199110	per706	7
39678	Switzerland	43110	199110	per706	9
39679	Switzerland	43951	199110	per706	6
39680	Switzerland	43320	199110	per706	12
39681	Switzerland	43710	199110	per706	0
39682	Switzerland	43520	199110	per706	8
39683	Switzerland	43321	199510	per706	5
39684	Switzerland	43110	199510	per706	3
39685	Switzerland	43810	199510	per706	4
39686	Switzerland	43320	199510	per706	14
39687	Switzerland	43520	199510	per706	5
39688	Switzerland	43530	199510	per706	0
39689	Switzerland	43420	199510	per706	4
39690	Switzerland	43110	199910	per706	34
39691	Switzerland	43520	199910	per706	14

	countryname	party	date	category	catcount
39692	Switzerland	43320	199910	per706	49
39693	Switzerland	43810	199910	per706	23
39694	Switzerland	43420	199910	per706	0
39695	Switzerland	43530	199910	per706	6
39696	Switzerland	43320	200310	per706	29
39697	Switzerland	43530	200310	per706	3
39698	Switzerland	43110	200310	per706	0
39699	Switzerland	43520	200310	per706	3
39700	Switzerland	43711	200310	per706	2
39701	Switzerland	43810	200310	per706	28
39702	Switzerland	43420	200310	per706	7
39703	Switzerland	43531	200310	per706	0
39704	Switzerland	43220	200310	per706	2

```
In [34]: cmpdataLong <- mutate(cmpdataLong, year = floor(date / 100))
cmpdataLong %>%
  group_by(category, year) %>%
  summarize(total = sum(catcount)) %>%
  filter(category %in% c("per108", "per110")) %>%
  ggplot(aes(x = year, y = total, colour = category, group = category)) + geom_line()
```





Long v. wide formats (Python)

- pandas: <https://pandas.pydata.org/pandas-docs/stable/reshaping.html>
(<https://pandas.pydata.org/pandas-docs/stable/reshaping.html>)

Upcoming

- **Lab:** Reshaping data in R and/or Python
- **Next week:** Creating and managing databases