Literary Data: Some Approaches

Andrew Goldstone
http://www.rci.rutgers.edu/~ag978/litdata

Thursday, January 22, 2015. Introduction.

Computing

High-level view

A program transforms inputs to outputs in a predictable way.

Computing

High-level view

A program transforms inputs to outputs in a predictable way.

Low-level view

1. Perform calculations with numbers.
2. Store results of those calculations.
3. Perform additional calculations on the basis of those results.

Components of the machine

1. Processor: follows instructions, one by one

Components of the machine

1. Processor: follows instructions, one by one
2. Memory: a series of numbered “addresses,” each one holding a fixed amount of data, which can be read or written by the processor

Components of the machine

1. Processor: follows instructions, one by one
2. Memory: a series of numbered "addresses," each one holding a fixed amount of data, which can be read or written by the processor
3. Program: a list of instructions for the processor

Components of the machine

1. Processor: follows instructions, one by one
2. Memory: a series of numbered "addresses," each one holding a fixed amount of data, which can be read or written by the processor
3. Program: a list of instructions for the processor
    - add, subtract, load, store…

Components of the machine

1. Processor: follows instructions, one by one
2. Memory: a series of numbered "addresses," each one holding a fixed amount of data, which can be read or written by the processor
3. Program: a list of instructions for the processor
    - add, subtract, load, store…
    - jump to the instruction numbered…

Components of the machine

1. Processor: follows instructions, one by one
2. Memory: a series of numbered "addresses," each one holding a fixed amount of data, which can be read or written by the processor
3. Program: a list of instructions for the processor
   - add, subtract, load, store…
   - jump to the instruction numbered…
   - jump *if* the following conditions hold…

Components of the machine

1. Processor: follows instructions, one by one
2. Memory: a series of numbered "addresses," each one holding a fixed amount of data, which can be read or written by the processor
3. Program: a list of instructions for the processor
   - add, subtract, load, store…
   - jump to the instruction numbered…
   - jump *if* the following conditions hold…
4. *The program lives in memory.*

Computer language

- a formally constrained way of specifying an algorithm
- translated into machine instructions by a program (input: formal description: output: sequence of machine codes), either an *interpreter* or a *compiler*
- a *high-level* language provides convenient *abstractions*

R: rings a Bell

1976–84: S language/environment developed at Bell Labs for statistical research, parallel to C and Unix projects

> The [language/environment] ambiguity is real and goes to a key objective: we wanted users to be able to begin in an interactive environment, where they did not consciously think of themselves as programming. Then as their needs became clearer and their sophistication increased, they should be able to slide gradually into programming, when the language and system aspects would become more important. This philosophy would be articulated explicitly later, but it was implicit from the start. (John Chambers)

1984–98: Commercial public-use versions of S (1984–98)

“R, also called GNU S”

1993–97: Ihaka and Gentleman develop open-source implementation, R
2000: R 1.0
2004: R 2.0
2013: R 3.0

Characterizing R

- interpreted
- functional
- object-oriented
- vectorized
- weakly typed
- kinda funky

The interpreted world

- console interaction
- *or* script execution

first steps in the console

R is a parrot

```
2
"Shiver me timbers"
```

R gets crabby easily

```
Shiver
Shiver me timbers
help
(
"Shiver
```

Press `esc`.

scripting: silly exercise

Enter:

```
2.7
print(2.7)
4 + 4
print(4 + 4)
"Stately, plump"
print("Stately, plump")
```

What does `print(...)` do?

(silly, cont.)

1. Copy and paste the statements without `print` into a new R script.
2. Click "Source."
3. Paste in the "print" statements. Click "Source" again.

What is going on?

human language/computer language

Try this in the console:

```
compute I say!
```

and this:

```
# compute I say!
```

human/computer

Create a new “R Markdown” file:

````
```{r}
2 + 2
```

I can say *anything* I want.

```{r}
print(2 + 2)
```

Inline: `r 16 * 16`.
````

Click “Knit PDF.” What is going on?

“literate programming”

I believe that the time is ripe for significantly better documentation of programs, and that we can best achieve this by considering programs to be *works of literature*….

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.

The practitioner of literate programming can be regarded as an essayist, whose main concern is with exposition and excellence of style.

(Knuth, “Literate Programming,” 1984)

“literate”

- Interleave discussion and program implementation
- “Knit” (orig. “weave”) the source into a finished, typeset output incorporating the results of program execution

“literate”

- Interleave discussion and program implementation
- “Knit” (orig. “weave”) the source into a finished, typeset output incorporating the results of program execution

- Well-suited to data analysis projects
- Ill-suited to interactive programs, systems programming…

the human part: markdown

The principle

Make plain text more expressive with some extra *conventions*.

- still pretty easy for a human writer/reader to interpret
- but systematic enough to be processed programmatically

text conventions

emphasis `*emphasis*` or `_emphasis_`

bold `**bold**`

typography: “curly” `("curly")`

and dashing: 1920–23 `1920--23`

and—this! `and---this!`

white space does matter

```
Paragraphs are broken by blank lines.

This is the start of a new paragraph.
But this isn't.

A backslash at the end of a line\
makes a "hard linebreak."
```

Paragraphs are broken by blank lines.

This is the start of a new paragraph. But this isn’t.

A backslash at the end of a line
makes a “hard linebreak.”

code

```
    Four spaces at the start of a line mark "code."
    This text is meant literally: *not styled*.
```

But to create executable R code, remember:

````
```{r}
# R code goes here
2 + 2
```
````

marking structure

```
# Heading
## Subheading
### Subsubheading

> A block quotation, which can be
> spread over multiple lines if you like.
```

Heading

Subheading

Subsubheading

> A block quotation, which can be spread over multiple lines if you like.

more markdown

Footnotes, URLs, lists… See http://rmarkdown.rstudio.com.

data types: simple: numerical

- Whole numbers (integer scale). *How many* (books, people, words, genres…)?
- Real numbers (interval scale). *How much* (distance, time, money…)? Special cases:
  - percentages or proportions (ratio scale). *How much of the total* (population, corpus of texts…)?
  - dates. *When?* (And does the day, month, year, decade, century… matter?)

data types: simple: categorical

- Unordered. *Which of…* (languages, nations, genders(?))? Special cases:
  - binary or *Boolean* category: true or false, yes or no.
  - many categories (headwords in the dictionary, authors in the catalogue).
- Ordinal. *Which* (letter of the alphabet, sales rank, "like, dislike, or neutral")?

Categories to numbers

- true: 1, false: 0
- like: 1, neutral: 0, dislike: -1
- like: 2, neutral: 1, dislike: 0
- a: 1, b: 2, c: 3… (character encoding)

data types: compound

The list / the series

18.2, 16.9, 17.5, 19.8, 21.9, 22.8, 23.5, 21.0, 18.1, 19.8

data types: compound

The list / the series

18.2, 16.9, 17.5, 19.8, 21.9, 22.8, 23.5, 21.0, 18.1, 19.8

(U.S.A. top 1% income share as percentage, 2001–2010
Piketty S8.2)

compound types, cont.

The list of lists / the table

```
firstname: Alice, Mo, Tomas
surname: Munro, Yan, Tranströmer
bornCountry: Canada, China, Sweden
```

```
firstname surname      bornCountry
Alice     Munro        Canada
Mo        Yan          China
Tomas     Tranströmer Sweden
```

(more elaborate possibilities exist…)

and text?

a (looooong) list of characters (a "string"):

```
O, n, c, e, *space*, u, p, o, n, *space*, a, *space*,
t, i, m, e
```

other representations

- the bag of words (to: 2, be: 2, or: 1, not: 1)
- content analysis (automated, human, or semi-automated)
- marked-up text

```
<sp who="#Salinus"><speaker>Duke.</speaker>
<p>Haplesse <name>Egeon</name> whom the fates
haue markt...</p>
```

- parsed trees
- page images

R formats for simple data

numeric	`12` or `1.618` or `6.02e23`
character	`"Whoops-a-daisy!"`
logical	`TRUE` or `FALSE`
	(or…`NA`)
factor	(categorical variables: later)

assignment

Storing values under names:

```
name <- value # read "let name be value"
```

```
n_students <- 9
degree_sought <- "Ph.D."
dh_is_a_coherent_theoretical_program <- FALSE
```

Choosing names

any combination of letters, numerals, `.` and `_`
(can't begin with a numeral)

suggested convention: `snake_case`

use/mention

a variable name evaluates to its current value

except on the left-hand side of `<-`

```
n_students
n_students * 2
n_students <- n_students + 1 # what is n_students now?
n_students <- (n_students + 4) / 2 # and now?
```

first compound data type: the vector

vector a finite sequence of data, all of the same type

first compound data type: the vector

vector a finite sequence of data, all of the same type

Instead of:

```
age1 <- 25
age2 <- 28
age3 <- 22
```

Write:

```
age <- c(25, 28, 22)
```

```
age[1] # read: "age sub 1"
```

```
[1] 25
```

```
age[2]
```

```
[1] 28
```

```
age[3]
```

```
[1] 22
```

```
age
```

```
[1] 25 28 22
```

```r
name1 <- "Adorno"
name2 <- "Benjamin"
name <- c("Adorno", "Benjamin") #  more dialectical
name
```

```
[1] "Adorno"   "Benjamin"
```

vector operations

A special vector expression:

```
1:4 # sequence
```

```
[1] 1 2 3 4
```

vector operations

A special vector expression:

```
1:4 # sequence
```

```
[1] 1 2 3 4
```

Concatenation:

```
x <- 1:3
y <- 3:5
c(x, y)
```

```
[1] 1 2 3 3 4 5
```

```
c(y, x)
```

```
[1] 3 4 5 1 2 3
```

How about:

```
c(x, x, y)
```

```
[1] 1 2 3 1 2 3 3 4 5
```

How is this consistent with defining vectors like `x <- c(1, 2, 3)`?

functions (first look)

A function maps inputs to outputs, possibly with some *side effects*.

Functions are said to be *applied* or *invoked*:

```
y <- f(x) # apply f to x, store result in y
result <- process_data(data1, data2, data3)
```

A function we know:

```
y <- c(1, 2) # apply c to 1 and 2, store result in y
```

functions (first look)

A function maps inputs to outputs, possibly with some *side effects*.

Functions are said to be *applied* or *invoked*:

```
y <- f(x) # apply f to x, store result in y
result <- process_data(data1, data2, data3)
```

A function we know:

```
y <- c(1, 2) # apply c to 1 and 2, store result in y
```

A new one:

```
y <- sqrt(16)     # sqrt maps a number to its square root
```

side effects

```
print(4)
```

On the homework: `setwd` and `getwd`

```r
# affects files on your hard drive as side-effect
install.packages("devtools")
# defines a whole bunch of functions as a side-effect
library("devtools")
# more file-grabbing
install_github("agoldst/litdata")
```

We've actually met one more function with side effects today. What is it?

next

- reading: Lévi-Strauss; McKenzie, 1–13; Darnton, 214–28 and nn.
- textbook: Jockers, chap. 1 (try whatever he says to try)
- optionally: Teetor, 2.1–2, 2.5–10, 2.13
- homework:
  www.rci.rutgers.edu/~ag978/litdata/hw1
  submit on Sakai

Doubts, troubles, concerns, random thoughts?

Always, always reach out!