Dealing with Large amounts of Scientific data

Gilles Tredan – M2ISS - 2017 session

Me

- Chargé de recherches CNRS.
- ≈8th time I teach this lecture.
- Computer Science background.
  - Distributed Systems
  - Networked Systems
  - *Focus:* Graphs, algorithms and metrology

gtredan@laas.fr

Us ! Outline of the Seminar

Starter	Intro (CDA,LDA,Big data)
Main Dish	R & Plotting
Dessert	Efficiency, Some ML Clustering/PCA/k-means. Dealing with Graphs.
More ?	Privacy, Transparency, Conclusion.

You ?

Let's start to collect data !

- How many use Linux ?
- How many use Latex ?
- How many use R ?
- How do you plot in general ?
  - No plots
  - Word/Excel/Open*
  - Gnuplot
  - Python/matplotlib ?
- Asymptotic complexity ?
- How many fingers do you type with ?

Disclaimer

The Lecture It's messy !

- Serendipitously fuzzy. Objective is to point rather than to demonstrate
- I value your feedback !
- If you don't understand something don't be afraid to ask

The lack of formalism

- Several concepts here are quite new, not yet formally defined.
- *It depends*: There is no golden rule everywhere
- Objective: foster discussion around problems arising with data processing.
- Provide "good practice" hints

Limits: Not much hadoop/pregel/Spark/NoSQL.

Introduction

- Big Data = Big Hype
- Digitalization of our daily lifes brought loads of data
- Cheap processing power
- Google showed a way to tranform data into money..

Big data, Data science, Science with data

Once we datafy things, we can transform their purpose and turn the information into new forms of value.

The Rise of Big Data By Kenneth Neil Cukier and Viktor Mayer-Schoenberger - FA May 2013

Europe and Scientific Data

Why the focus on scientific data? For starters, science is a cause of this data wave. Scientific discovery led to the microprocessors, optical fibres and storage media with which we create, move and store the data. And the continuing process of scientific discovery - in all disciplines from astronomy to economics - is generating a growing share of that new data. In one day, a high-throughput DNA-sequencing machine can read about 26 billion characters of the human genetic code. That translates into 9 terabytes - or 9 trillion data units - in the course of one year; alongside it is a wealth of related information that can be 20 times more voluminous. The total data flow: more than 20 new US Libraries of Congress each and every year. That is from one specialised instrument, in one scientific sub-discipline; enlarge that picture across all of science, across the world, and you start to see the dimension of the opportunity and challenge presented.

Most importantly, however, our focus is on scientific data because, when the information is so abundant, the very nature of research starts to change.

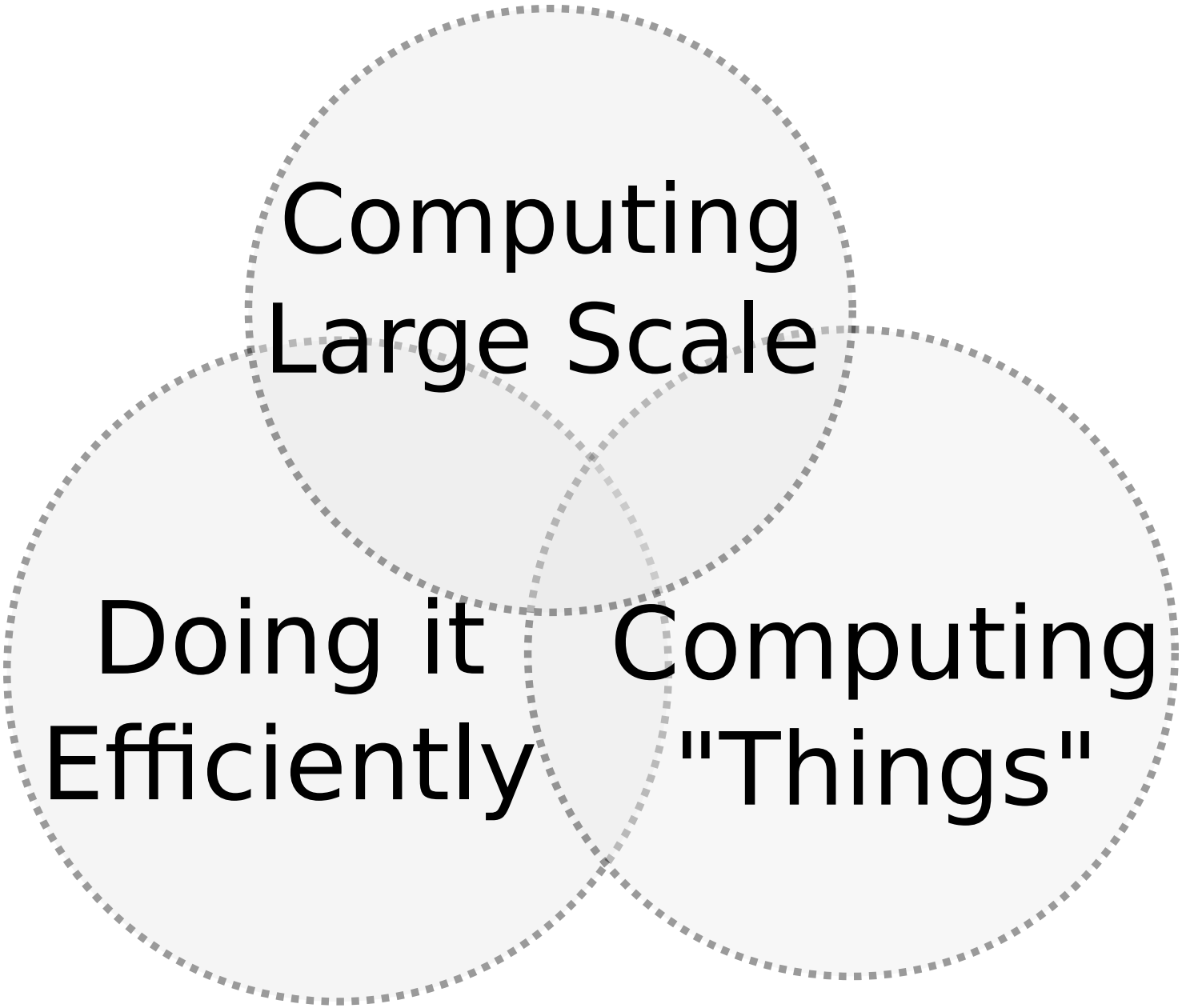
Final report of the High Level Expert Group on Scientific Data - H2020/European Comission - Oct. 2010

Specificities of Scientific Data Processing

My view !

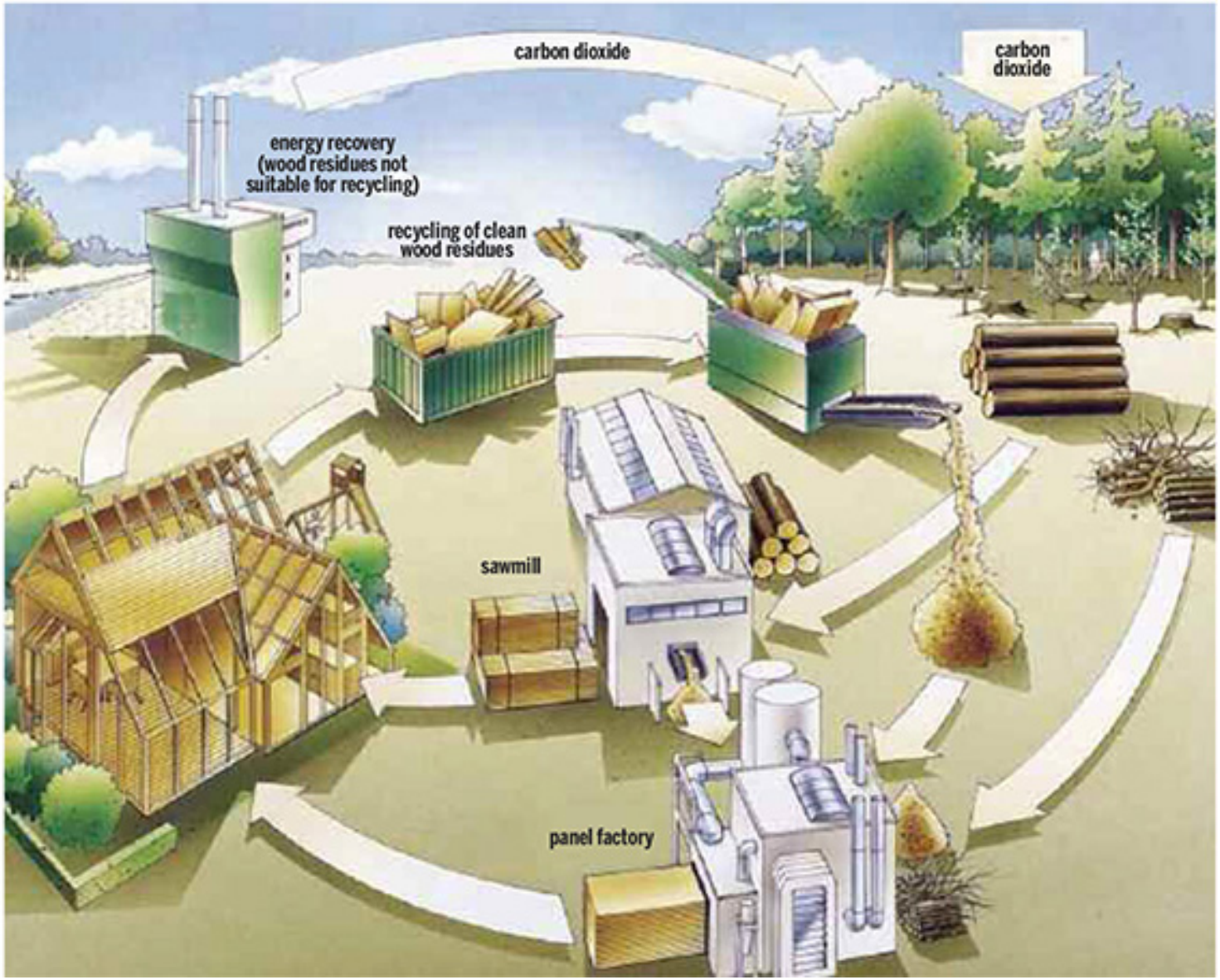
- **Cheaper**: most of the analyses are at the "system level"... No need to produce per-individual results, like recommendation (which is user based by definition) rather means for the hole population. A paper $\approx$ 5 figures
- **More experimental**: Analysis goals are often redefined or/and amended during the "exploratory phase" –new questions raise as we understand the data. Big data metrics are more absolute ($, recall) $\Rightarrow \neq$ coding process.
- Scientific processing of the data should **remain easily explainable** (for the target community) and rather "self-contained".
- **No** (pseudo) **real-time** constraints. Complexity is of course an issue, but less pressure.
- **More Aesthetic considerations**:
  Objective of a plot = convince ($\Rightarrow$look nice)

Big Data Approaches

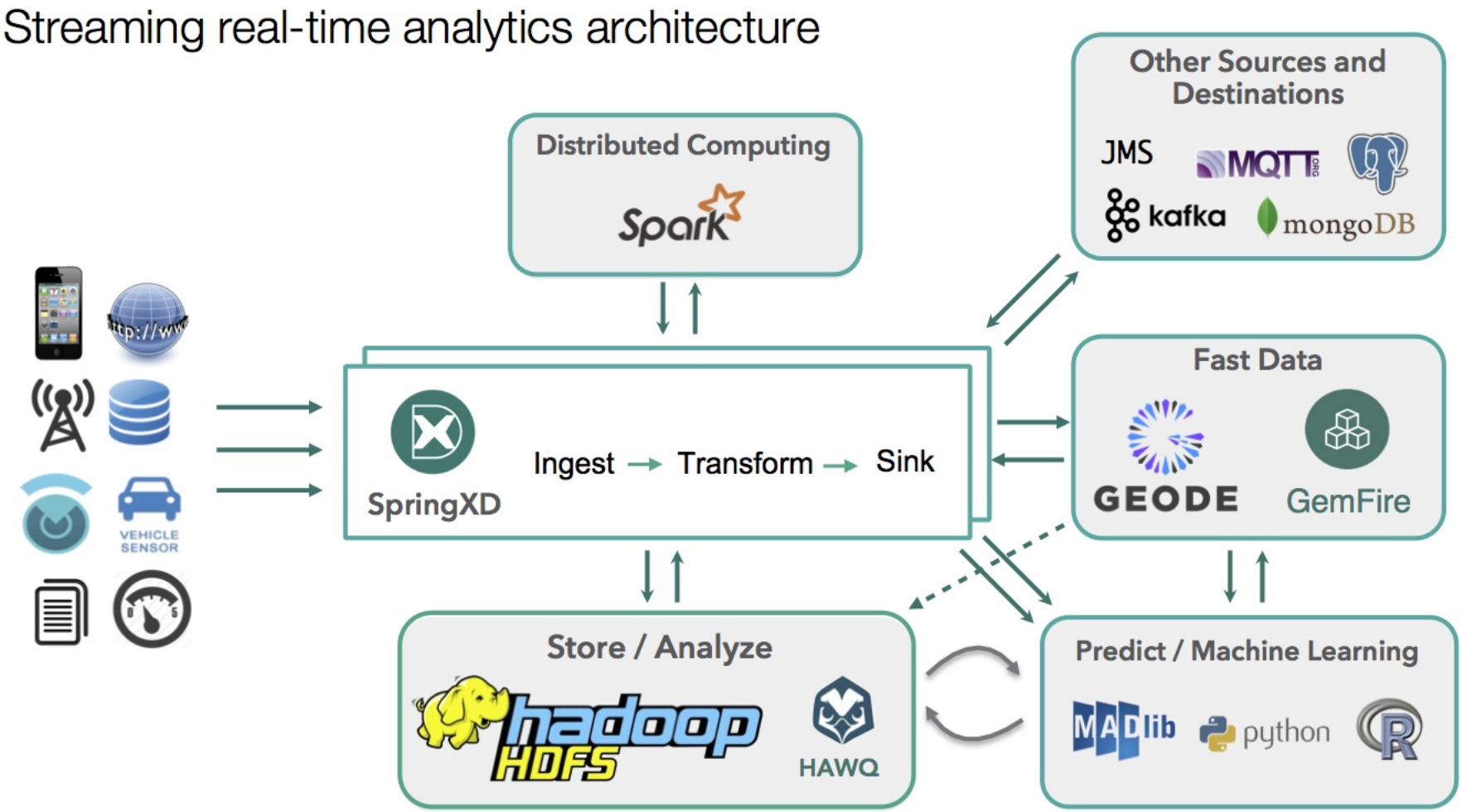


Objective of This Lecture

Size Matters

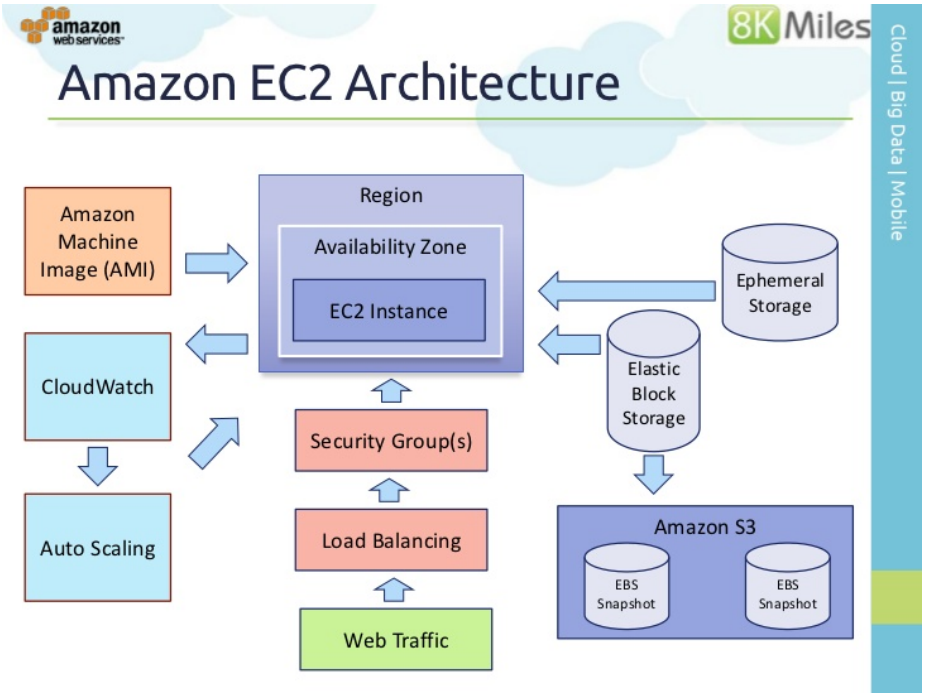


Size Matters



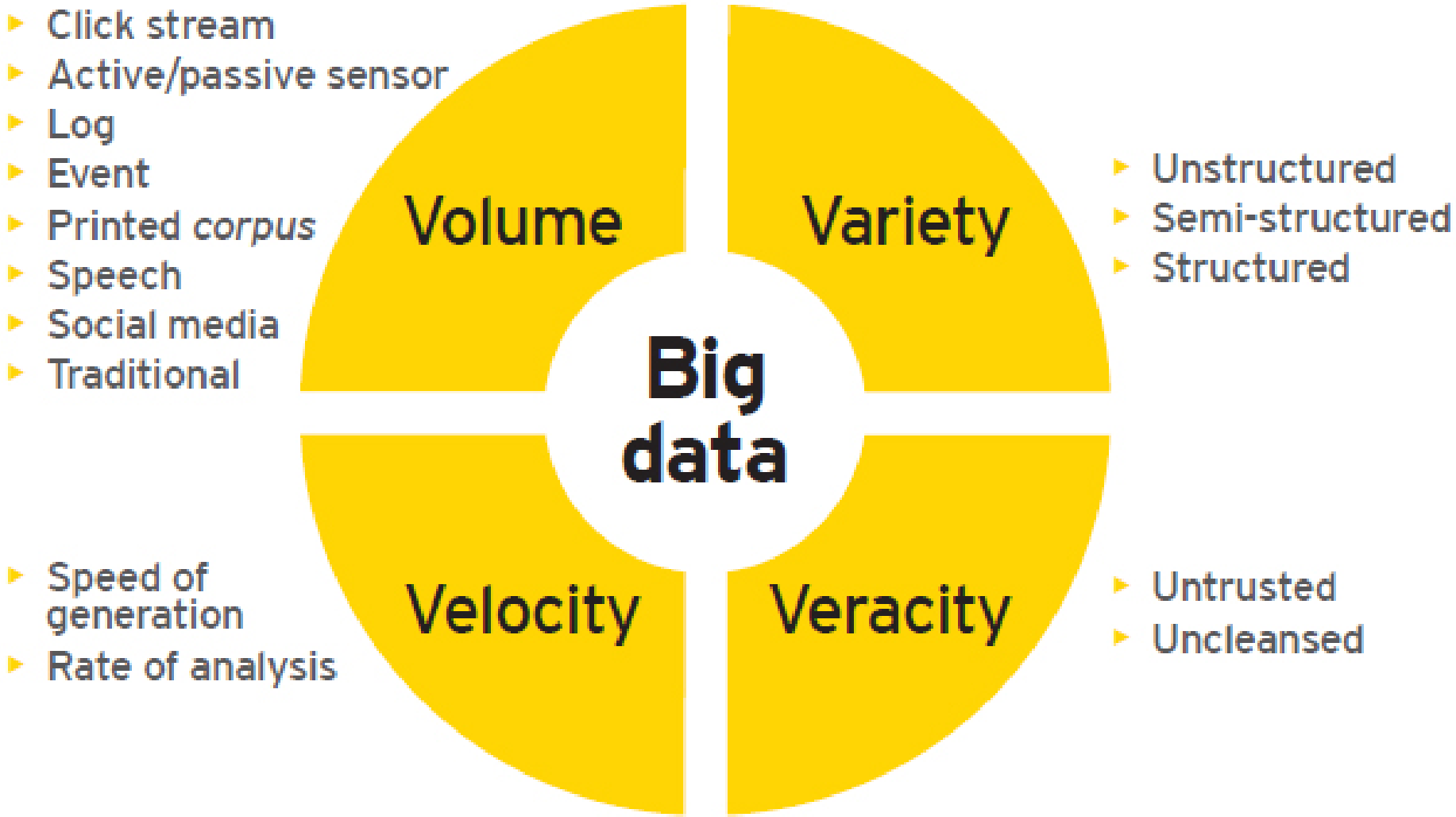
Size Matters

- Biggest datacenter= $92000m^2$
- Failure is the norm



- How to store
- How to power
- How to balance load
- How to synchronize
- How to upgrade

The 4 v's



Stolen from `http://www.wordlypost.in/big-data-new-way-understanding-world/`

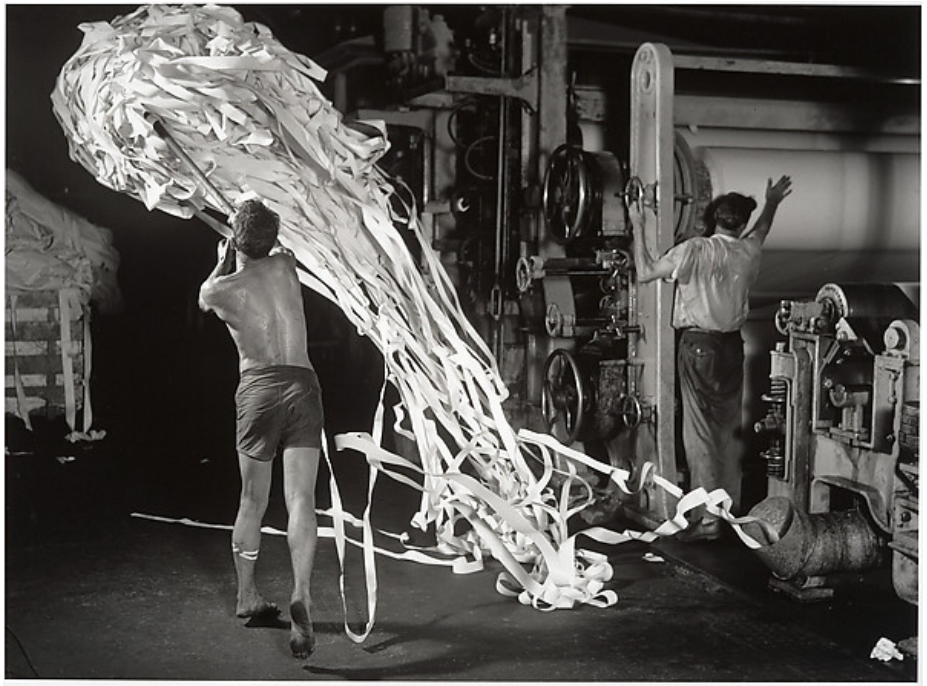
An arti perspective

left:Erro Foodscape, 1964 Oil on canvas, right: Wayne Thiebaud Salads, Sandwiches, and Desserts, 1962 Oil on canva Image from
http://novyden.blogspot.fr/2017/05/illustration-of-mapreduce-in-two-modern.html

Typical usecase

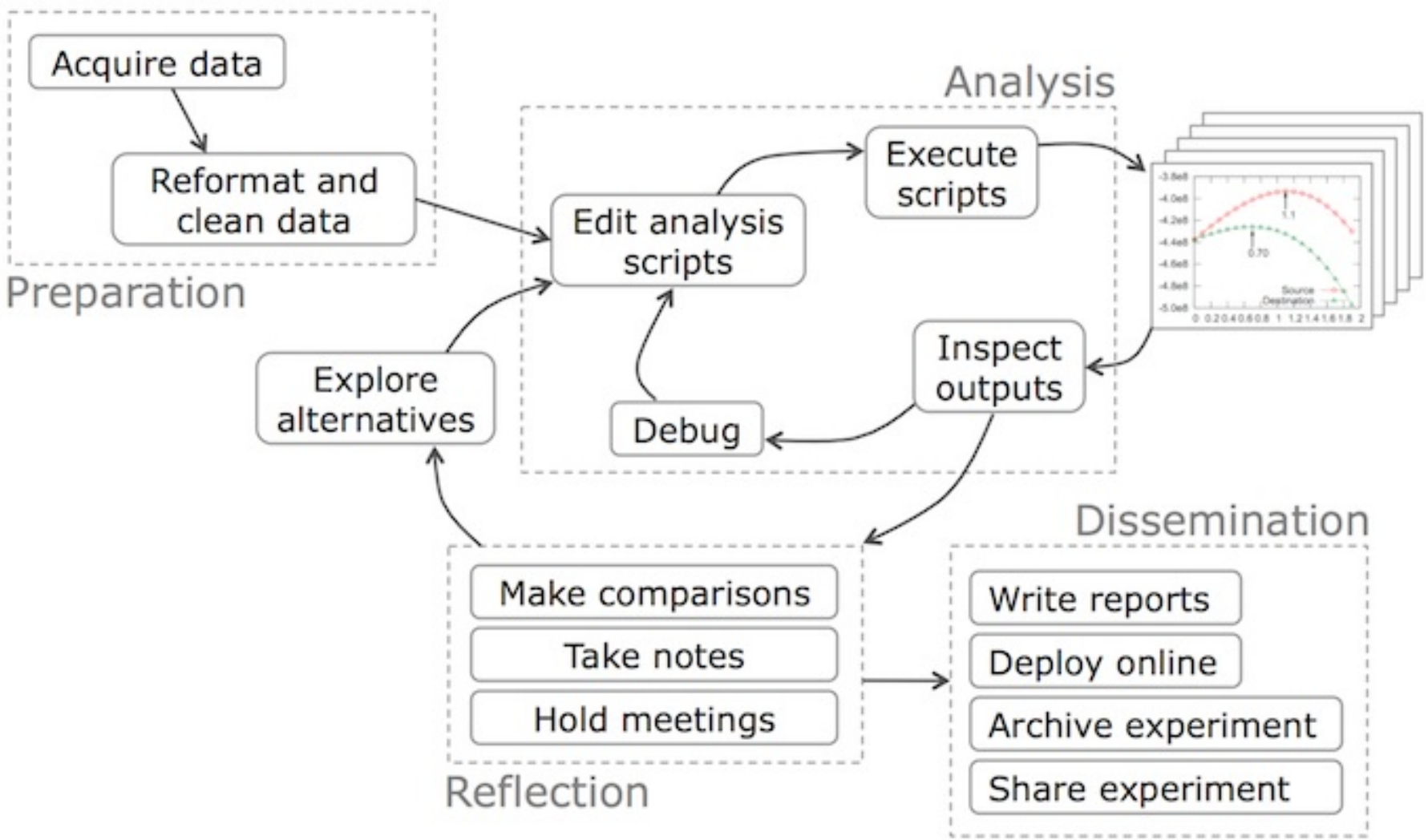
Actually my case

- Algorithm/ prototype/ hypothesis to test
- Simulate/capture data about this object's behaviour
- Want to understand/confirm this behaviour
- We want to predict/model this behaviour
- Evaluate the impact of parameters/exterior conditions



Harold Edgerton, 1937

Data Analysis Workflow



Stolen from
https://cacm.acm.org/blogs/blog-cacm/169199-data-science-workflow-overview-and-challenges/fulltext

Mistakes in Data Analysis

Three common mistakes among young scientists are assuming that:

- Data analysis occurs only after you are done collecting all your data.
- Data analysis is quick you pick your analysis methods, apply them in a "plug-in" fashion, and then you are done.
- Data can stand alone without additional context.
- *Analysing data is rewarding, and stands for a proof*

2 Sided Laziness:

- Think before you plot
- Don't plot to confirm

Cheap philosphy

If you collect data, you're interested in confronting to the real world.

- Bounded, error-prone capture
- Yet high-dimensional complex data
- Combined and Projected into apprehendable realities
- No neutrality but deontology.

statistics

Statisticians

Statistik (German,1749) "Science of the state"

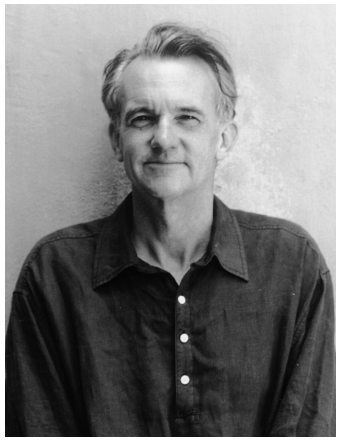
Tukey: exploratory data analysis $\neq$ confirmatory data analysis

- statistical methodology placed too great an emphasis on the latter
- FFT algorithm, box plot, the Tukey range test, the Tukey lambda distribution, the Tukey test of additivity...
- "bit" as a contraction of "binary digit"

Tufte: "Beautiful Evidence"

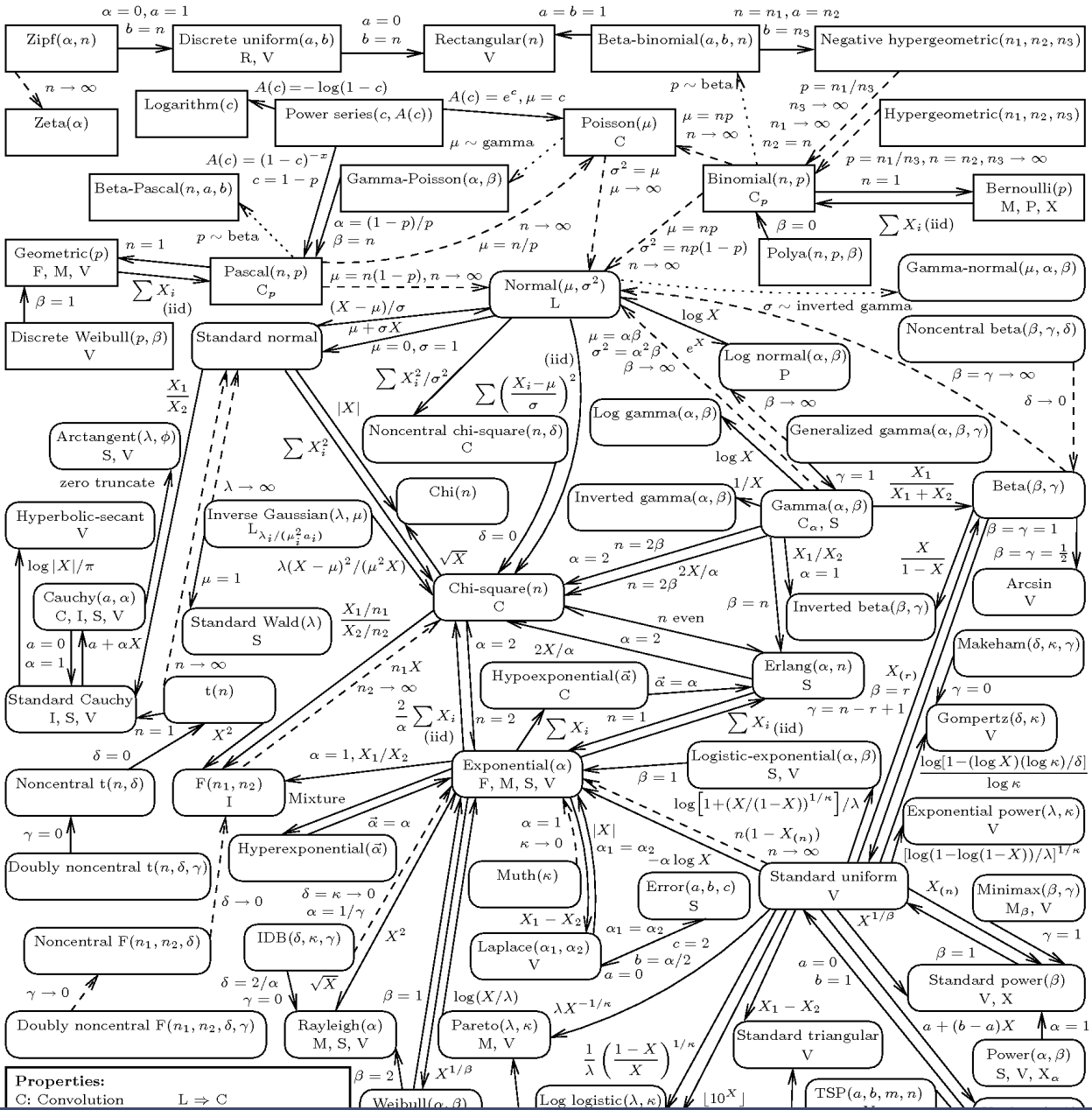
- "Formalised" the visual display of quantitative information.
- More later

John Tukey
(1915-2000)

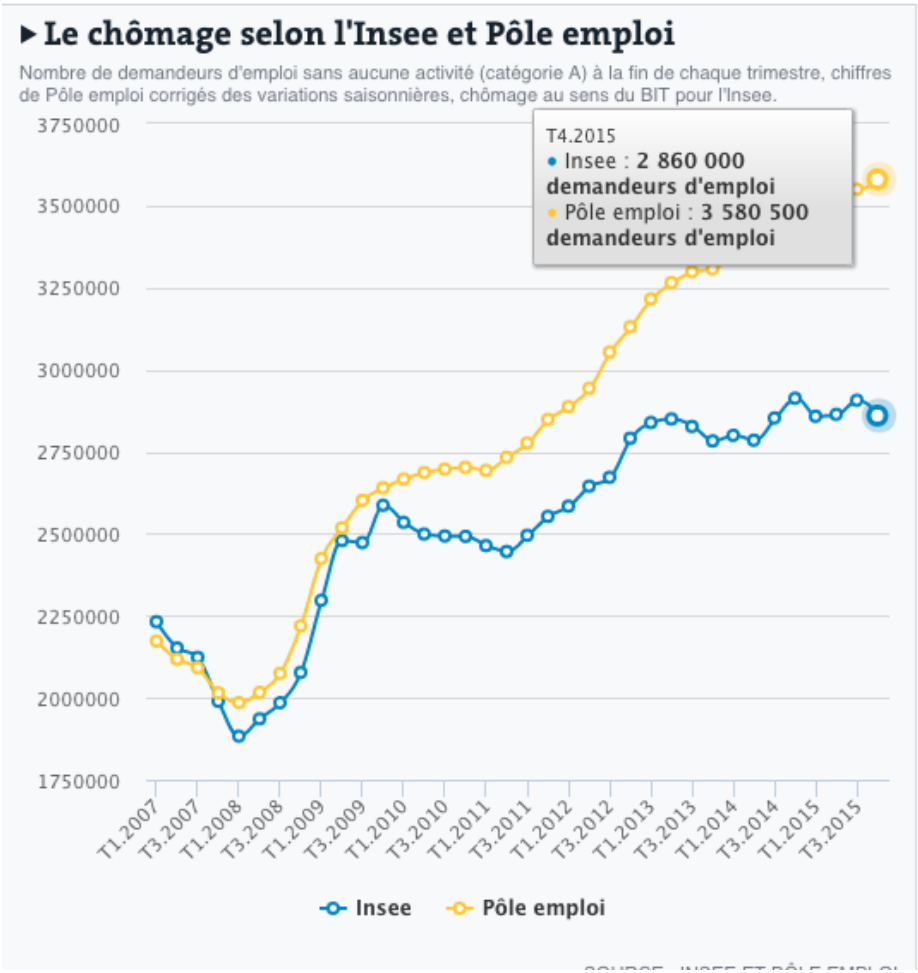


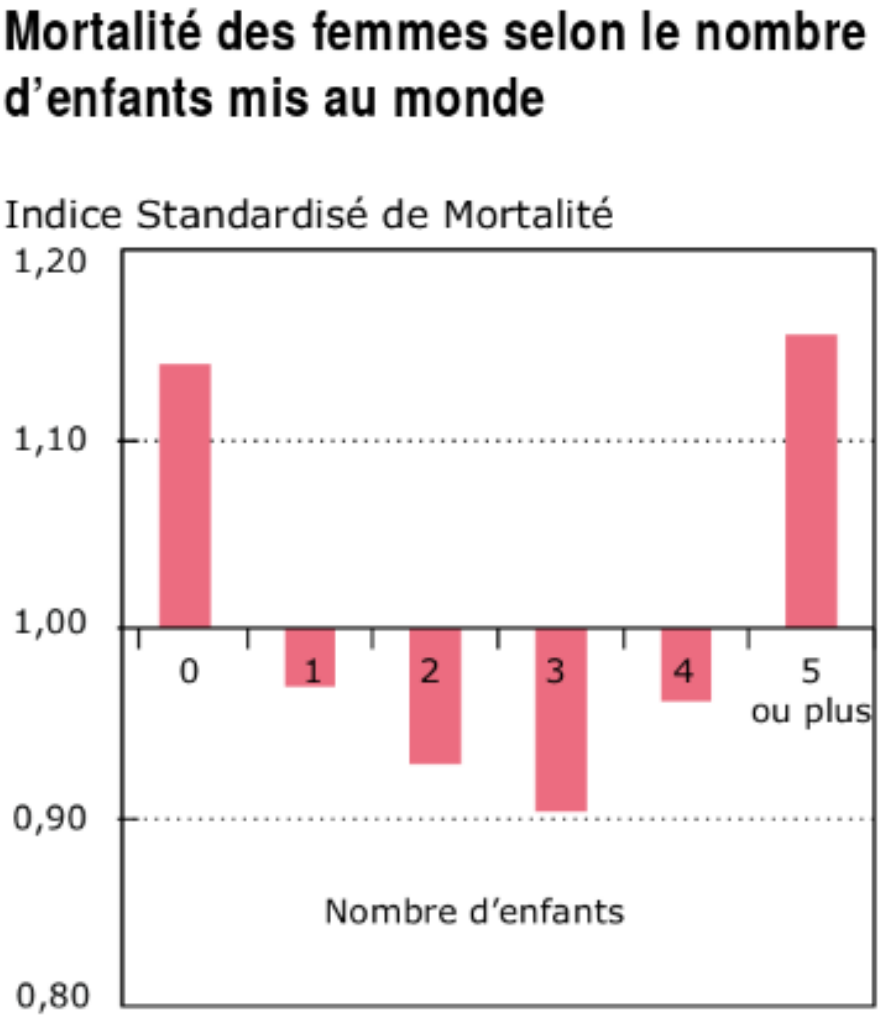
Edward Tufte
(1942-?)

Statistics are Complex



Statistics are Dangerous





y=f(x)

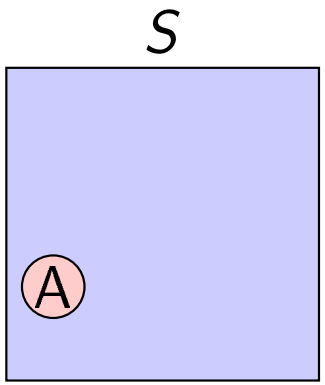
- What are y and x
  - random variables
  - qualitative (categorial) /quantitative/other
  - hard to measure and of interest ?
  - Difference= fundamental in the choice of tools (regression/classification for ML, histograms/venn diagrams/plots for graphics
  - limited of infinite support ! (difference=also fundamental)

  3 perspectives
  - ML= finding efficiently approximations of f
  - CDA= charting possible fs, studying them (e.g. bayes, $p(y|x)$)
  - EDA= choosing x,y and f couples

Random Variables

Event, Space, Probability

- An event A= anything
- Space S = "number of possible universes"
- Probability of A ($Pr(A)$) = number of universes in which A happens



Kolmogorov: Probability Axioms

- $0 \leq Pr(A) \leq 1$
- $Pr(S) = 1, Pr(\emptyset) = 0$
- $Pr(A \cup B) = Pr(A) + Pr(B) - Pr(A \setminus B)$

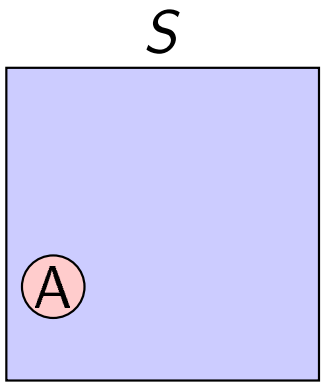
Discrete random variable: partitioning of S = "outcomes".

- $S = \{a_1, \ldots a_d\}$
- $Pr[X \in A \subseteq S] = \sum_{a_i \in A} Pr[X = a_i]$

Random Variables

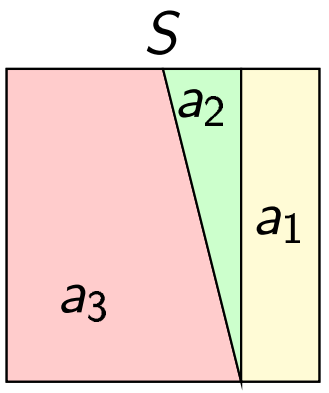
Event, Space, Probability

- An event A= anything
- Space S = "number of possible universes"
- Probability of A ($Pr(A)$) = number of universes in which A happens



Kolmogorov: Probability Axioms

- $0 \leq Pr(A) \leq 1$
- $Pr(S) = 1, Pr(\emptyset) = 0$
- $Pr(A \cup B) = Pr(A) + Pr(B) - Pr(A \setminus B)$



Discrete random variable: partitioning of S = "outcomes".

- $S = \{a_1, \ldots a_d\}$
- $Pr[X \in A \subseteq S] = \sum_{a_i \in A} Pr[X = a_i]$

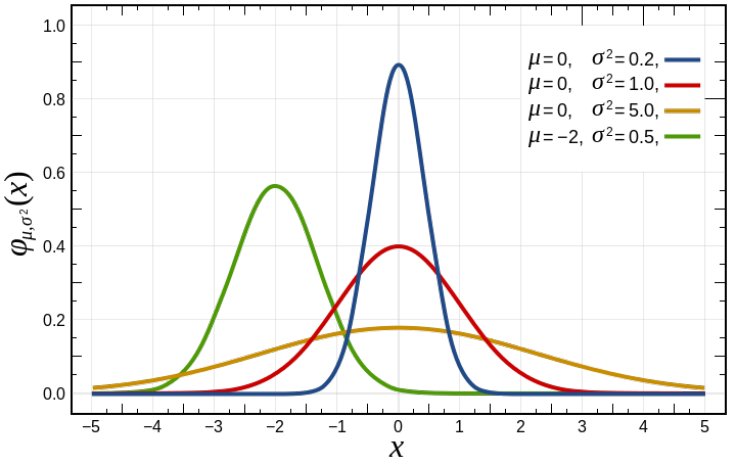
Thank you Mr. Ihler

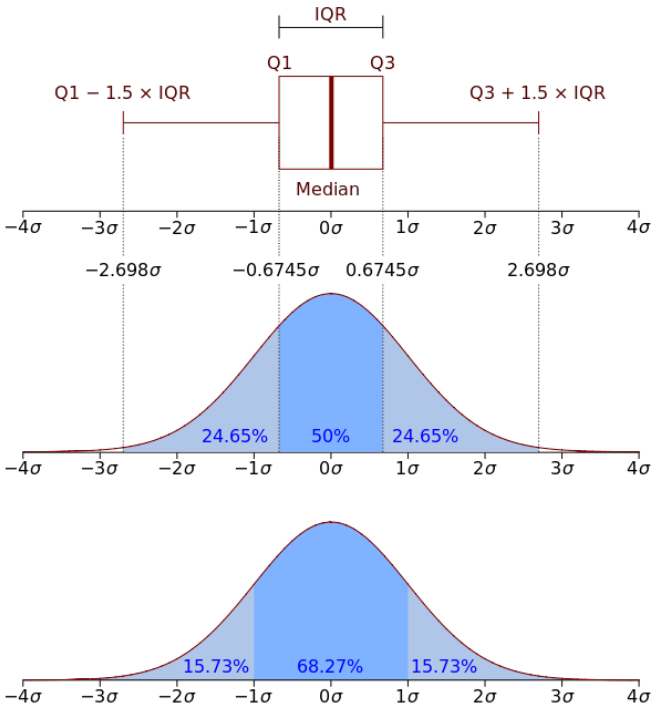
https://www.youtube.com/watch?v=8kmkF02lBlQ

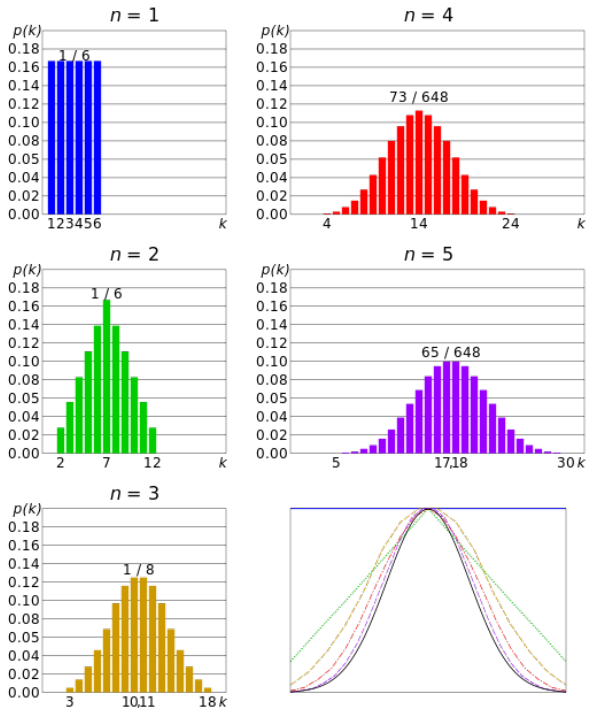
Normal Distribution

- Most important distribution
- Arises from the central limit theorem
- =smoothing effect of summation

$$f(x|\mu,\sigma) = \frac{1}{\sigma\sqrt{2\pi}}\, e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$



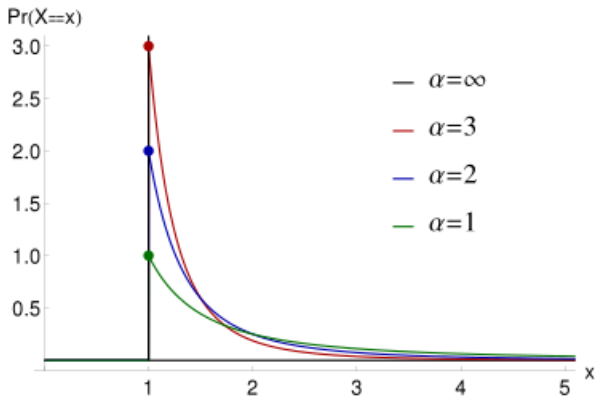




Not Normal distributions !

Pareto

- E.g. Recursive Pareto principle
- Everywhere in the world: Income, city sizes, oil reserves
- Size of sand particles
- ="Powerlaw"
- Heavy Tail !



$$\overline{F}(x) = \Pr(X > x) = \begin{cases} \left(\frac{x_{\mathrm{m}}}{x}\right)^{\alpha} & x \geq x_{\mathrm{m}}, \\ 1 & x < x_{\mathrm{m}}. \end{cases}$$

Exploratory vs Confirmatory data analysis

Exploratory data analysis (EDA) = explore the data first, identify patterns and outliers, and conclude.

- Suggest hypotheses about the causes of observed phenomena
- Assess assumptions on which statistical inference will be based
- Support the selection of appropriate statistical tools and techniques
- Provide a basis for further data collection through surveys or experiments

Confirmatory data analysis: A statistical hypothesis is a scientific hypothesis that is testable on the basis of observing a process that is modeled via a set of random variables. A statistical hypothesis test is a method of statistical inference used for testing a statistical hypothesis.

John W. Tukey, "Exploratory Data Analysis", 1977: Do not apply both on the same data set !

Confirmatory vs Exploratory Data Analysis

Exploratory Analysis

- **Descriptive Statistics - Inductive Approach**
  - Look for flexible ways to examine data without preconceptions
  - Attempt to evaluate validity of assumptions
  - Heavy reliance on graphical displays
  - Let data suggest questions
  - Focus on indications and approximate error magnitudes

- **Advantages**
  - Flexible ways to generate hypotheses
  - More realistic statements of accuracy
  - Does not require more than data can support
  - Promotes deeper understanding of processes
  - Statistical learning
- **Disadvantages**
  - Usually does not provide definitive answers
  - Difficult to avoid optimistic bias produced by overfitting
  - Requires judgement and artistry - can't be cookbooked

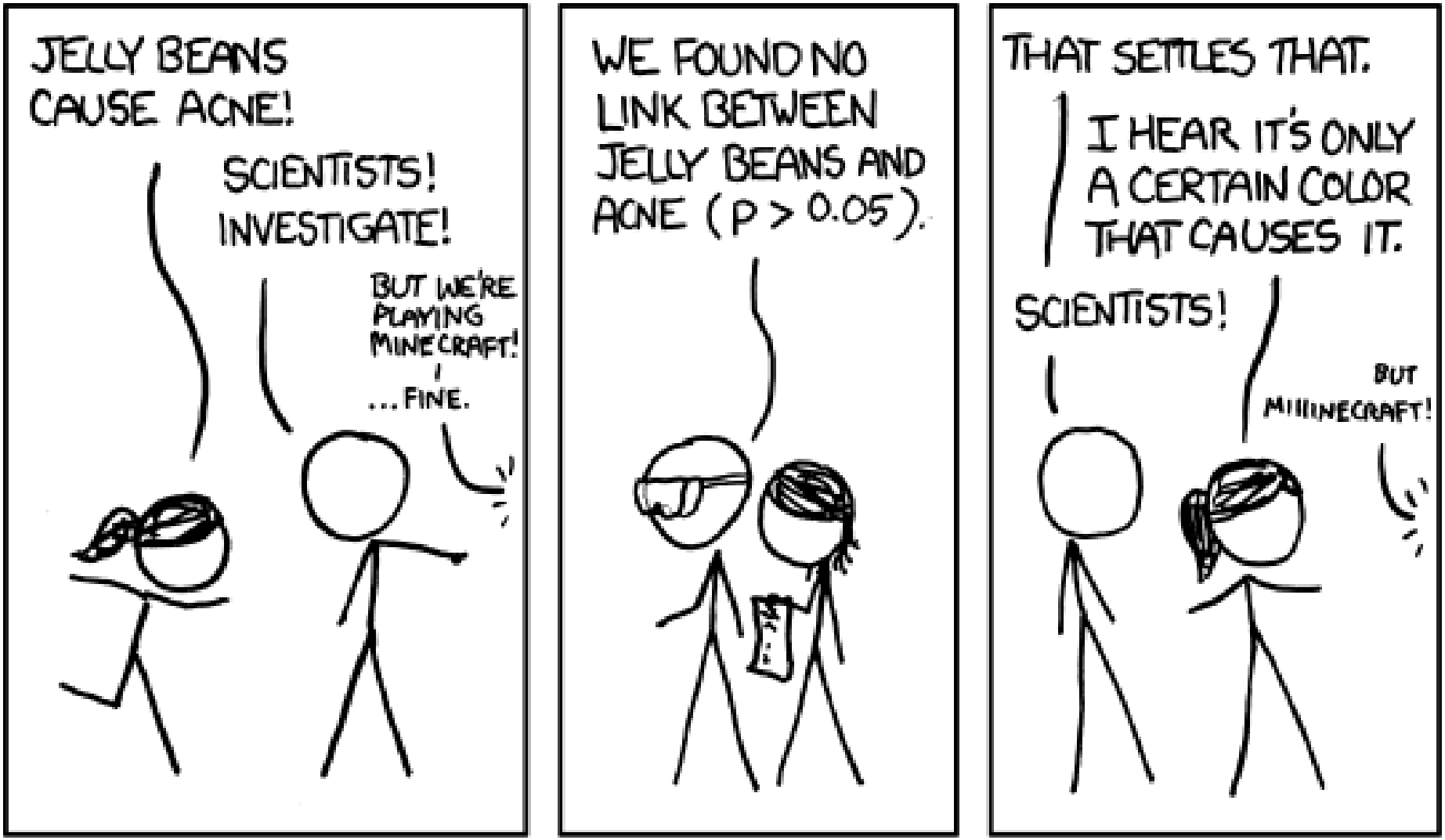
Confirmatory vs Exploratory Data Analysis

Confirmatory Analysis

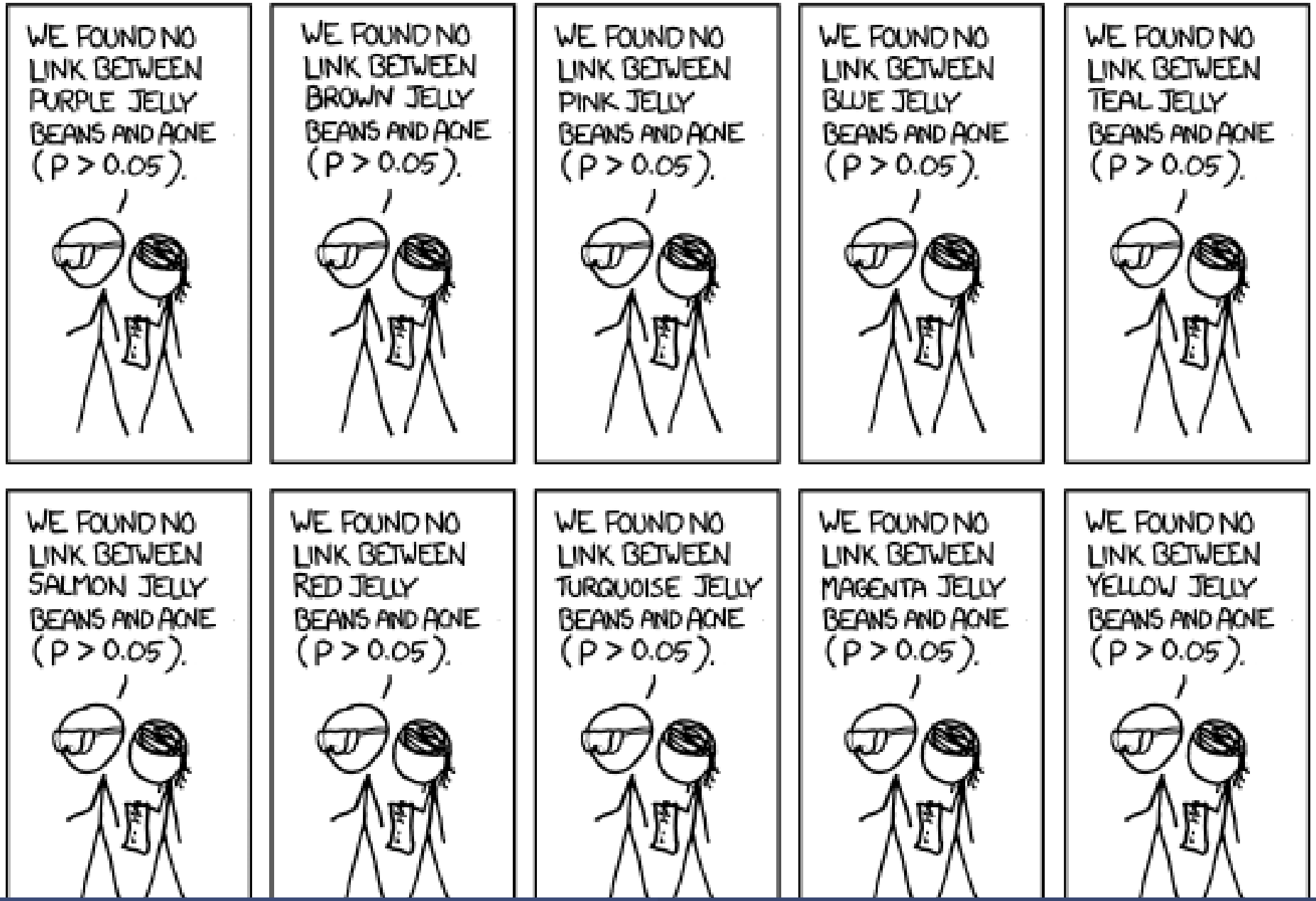
- **Inferential Statistics - Deductive Approach**
  - Heavy reliance on probability models
  - Must accept untestable assumptions
  - Look for definite answers to specific questions
  - Emphasis on numerical calculations
  - Hypothesis tests and formal confidence interval estimation

- **Advantages**
  - Provide precise information in the right circumstances
  - Well-established theory and methods
- **Disadvantages**
  - Misleading impression of precision in less than ideal circumstances
  - Analysis driven by preconceived ideas
  - Difficult to notice unexpected results

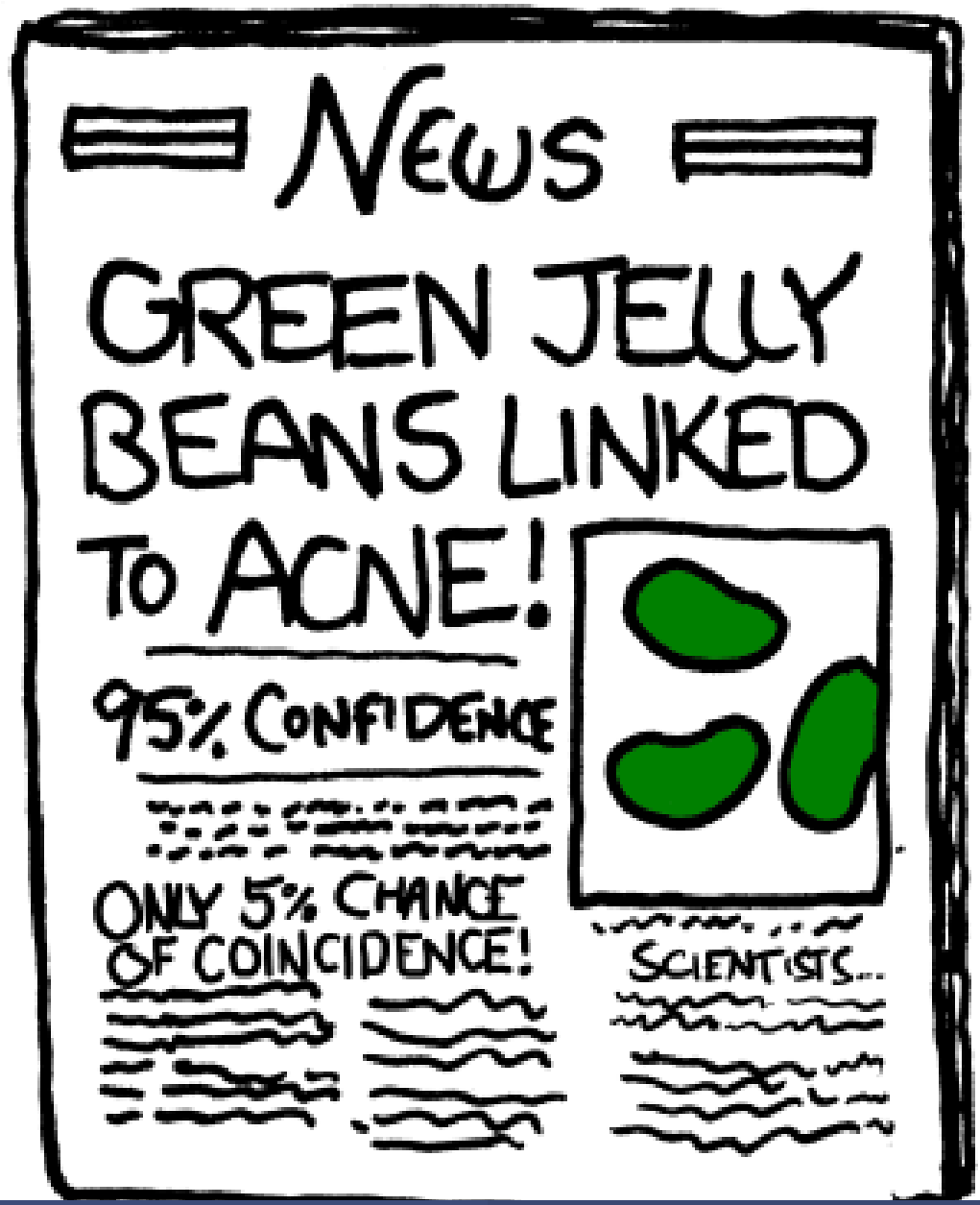
A word about p-value



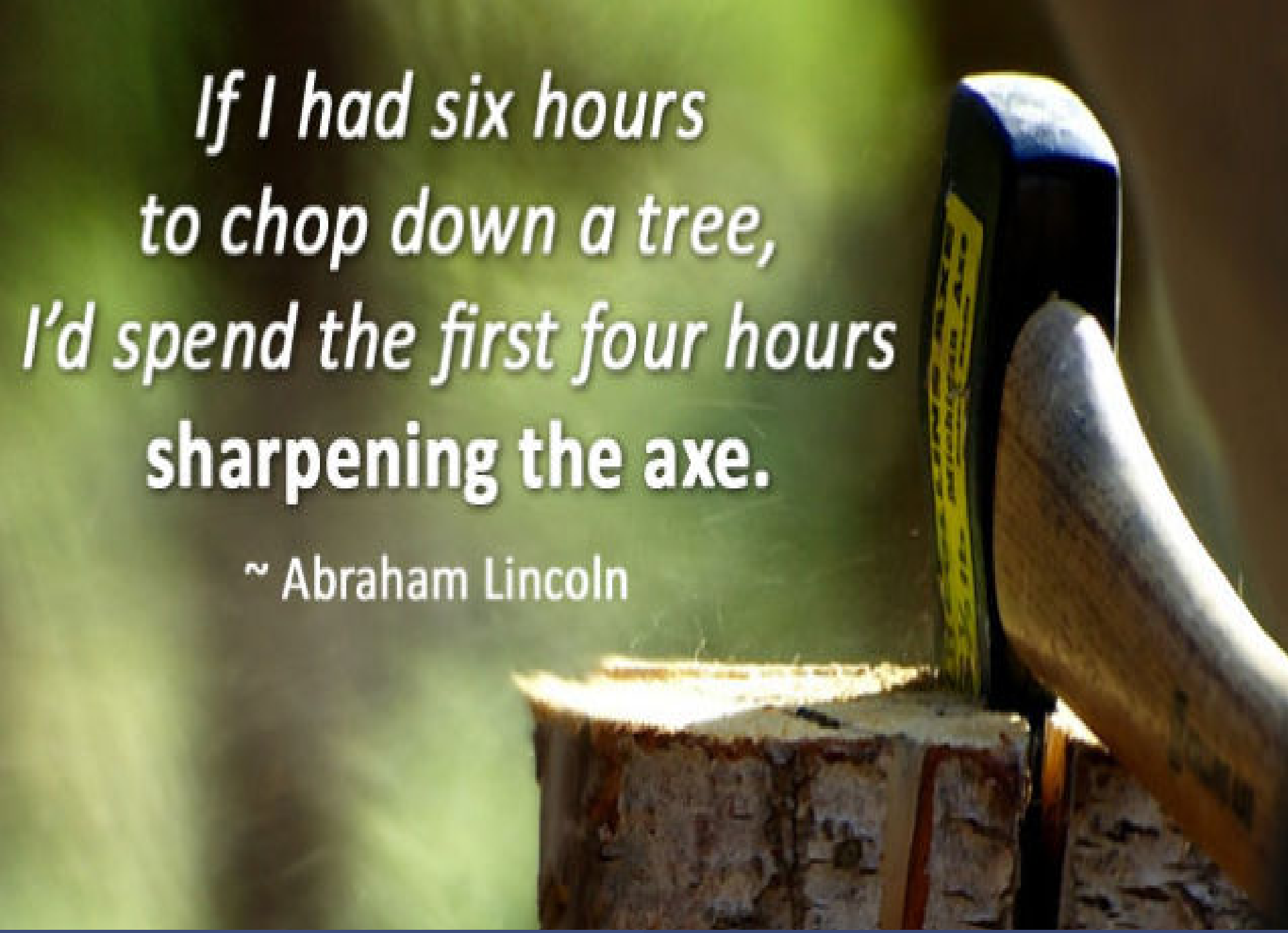
A word about p-value



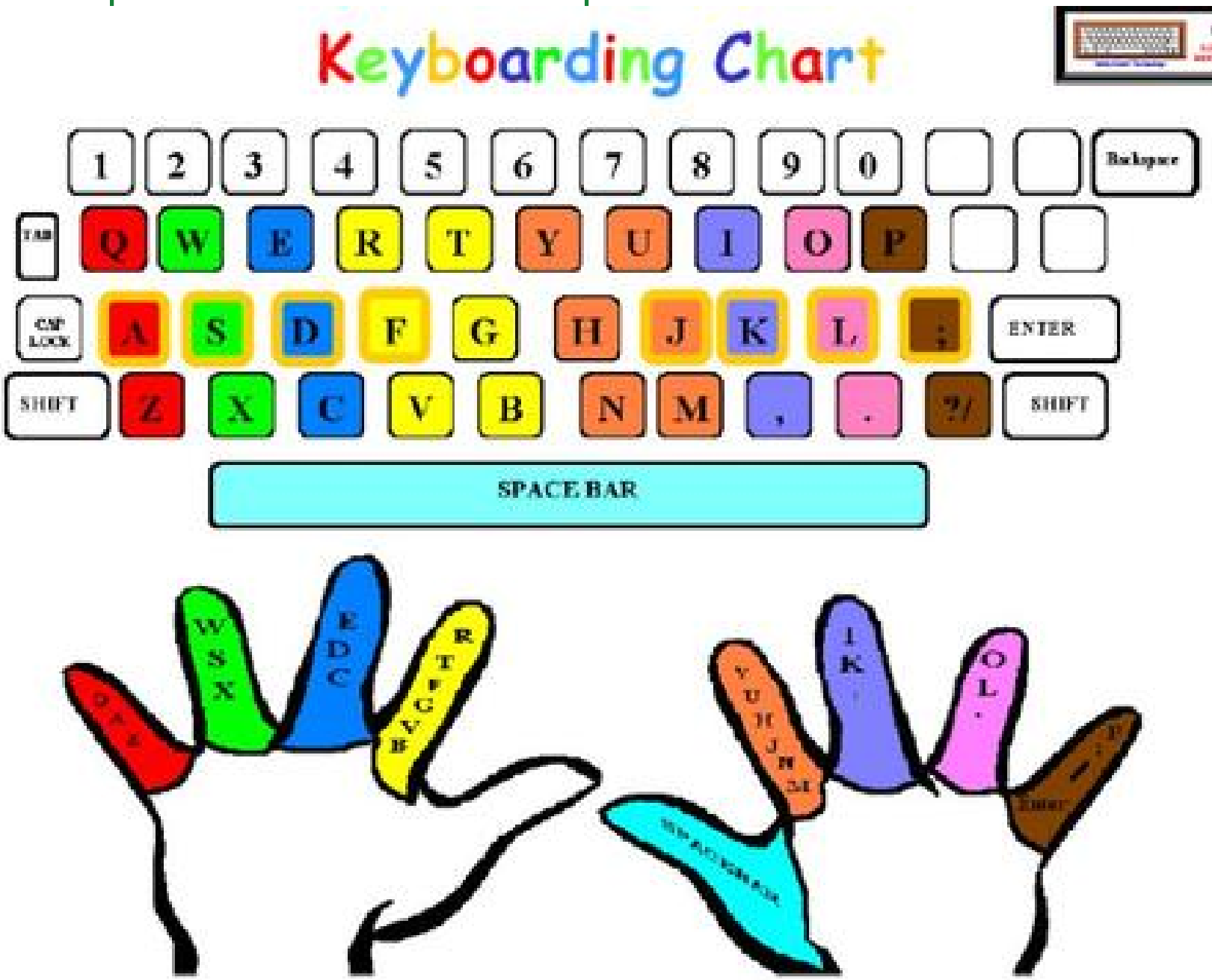
A word about p-value



General Considerations for Human Computer Collaboration


If I had six hours
to chop down a tree,
I'd spend the first four hours
sharpening the axe.
~ Abraham Lincoln

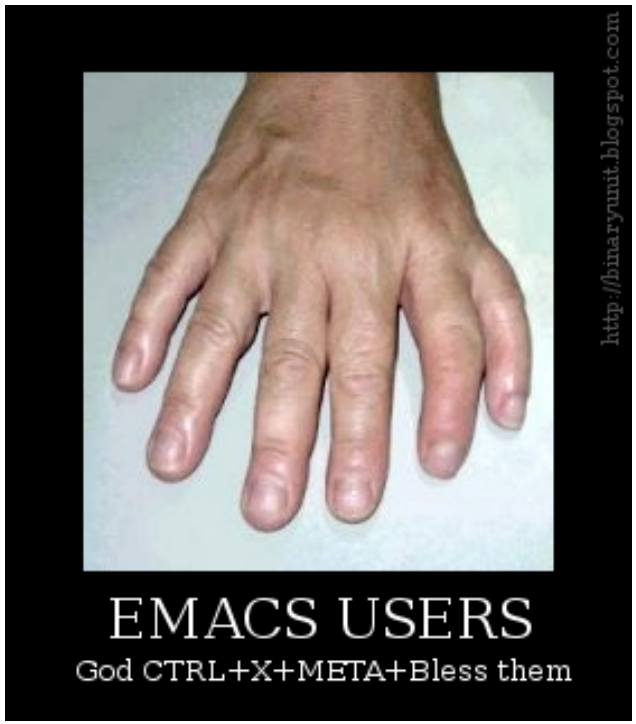
You will spend time on a computer



You will spend time writing text

Learn about your editor !

- Being efficient with an editor= spending time with it
- That's worth it !
- In this class I'll use emacs (with ess for R)
- Use what you feel efficient with !
- But don't use the mouse !

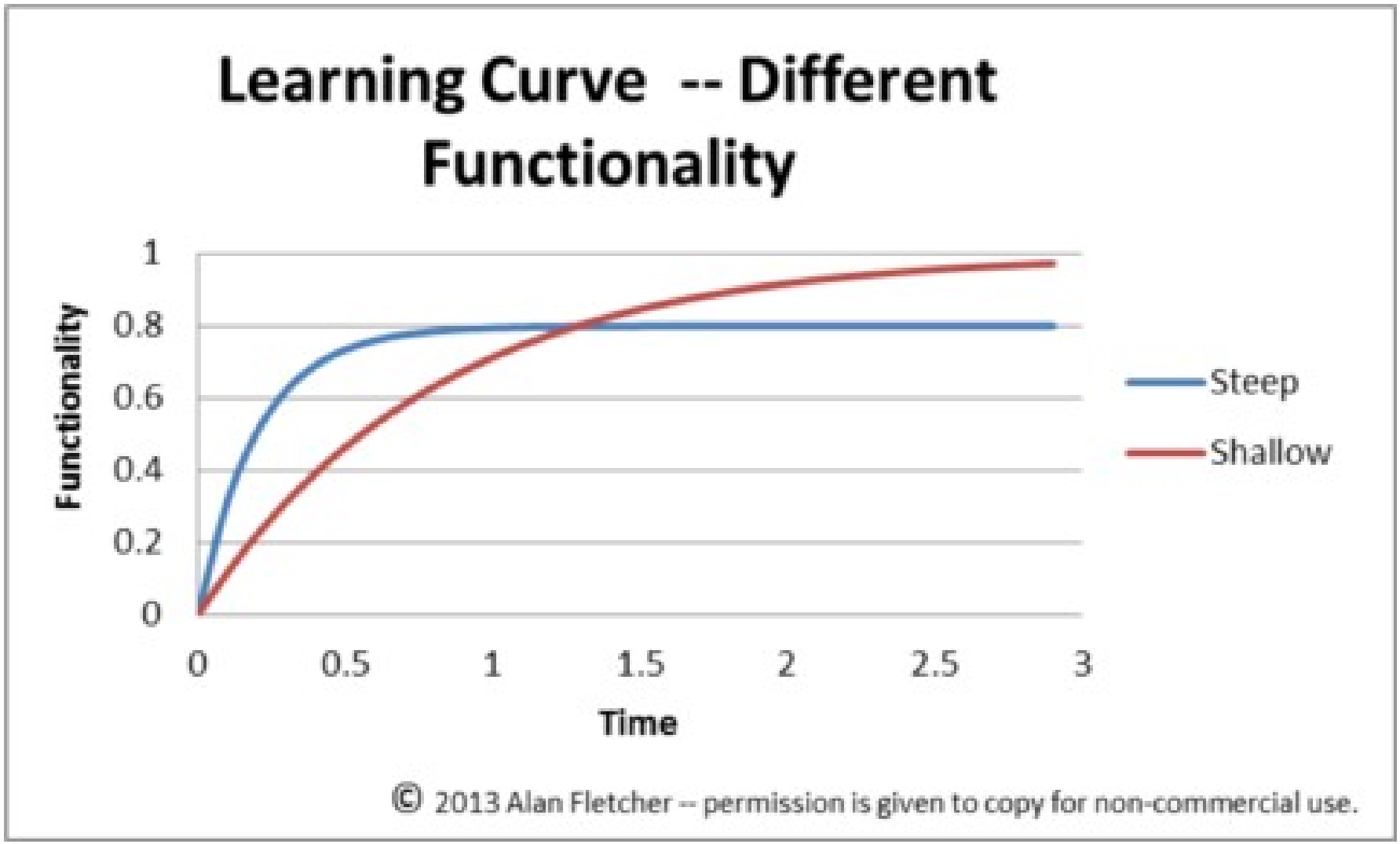


Emacs in some keystrokes

	Files, Buffers, Windows
C-x C-f	Find file (Open)
C-x C-s	Save buffer (Save)
C-x C-w	Write file (Save As)
C-x 2	Split window
C-x 1	Delete other windows (Unsplit)
C-x o	Other window
C-x C-b	Buffer menu
	Search and replace
C-s	Incremental search string forwards
C-r	Incremental search string backwards
M-%	Query replace string
C-M-s	Incremental search regexp forwards
C-M-r	Incremental search regexp backwards
C-M-%	Query replace regexp

	Lines
C-a	Beginning of line
C-e	End of line
C-k	Kill line (cut line)
	Words
M-b	Backwards word
M-f	Forwards word
M-d	Kill word (cut word)
	Region (between mark and point)
C-SPC	Set mark
C-w	Kill region (cut)
M-w	Copy region (copy)
	Pasting, Undoing
C-y	Yank (paste)
C-y M-y	Yank older copy (M-y)
C-_	Undo
	Rectangles
C-x r k	Kill rectangle
C-x r y	Yank rectangle

Persist !


Learning Curve -- Different Functionality
Functionality
1
0.8
0.6
0.4
0.2
0
0
0.5
1
1.5
2
2.5
3
Time
Steep
Shallow
© 2013 Alan Fletcher -- permission is given to copy for non-commercial use.

RTMF and LMGFY

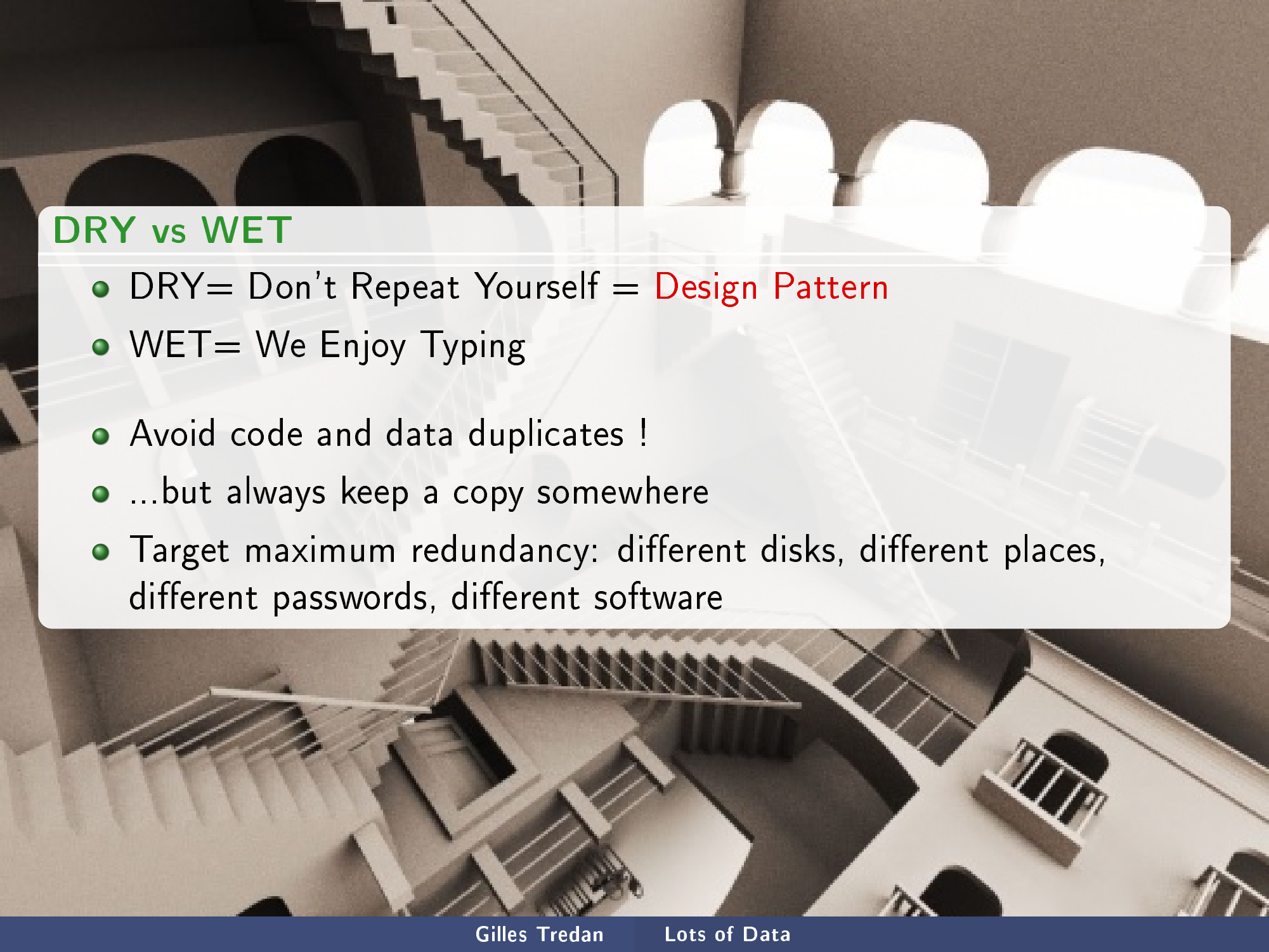
RTMF = Read the Fucking Manual
LMGFY = Let Me Google That For You !

- all the tools we will see have a common point: they are widely used.
- $\Rightarrow$tons of information on the web, examples/tutorials/manuals...
- $\Rightarrow$harness this information !
- Learning how to seek information in these sources is a must !

Be kind to you!

- Help your future self
- Use informative names
- Organise your directories
- Comment and avoid ugly construction, magic numbers, special tricks...

`https://google-styleguide.googlecode.com/svn/trunk/Rguide.xml`

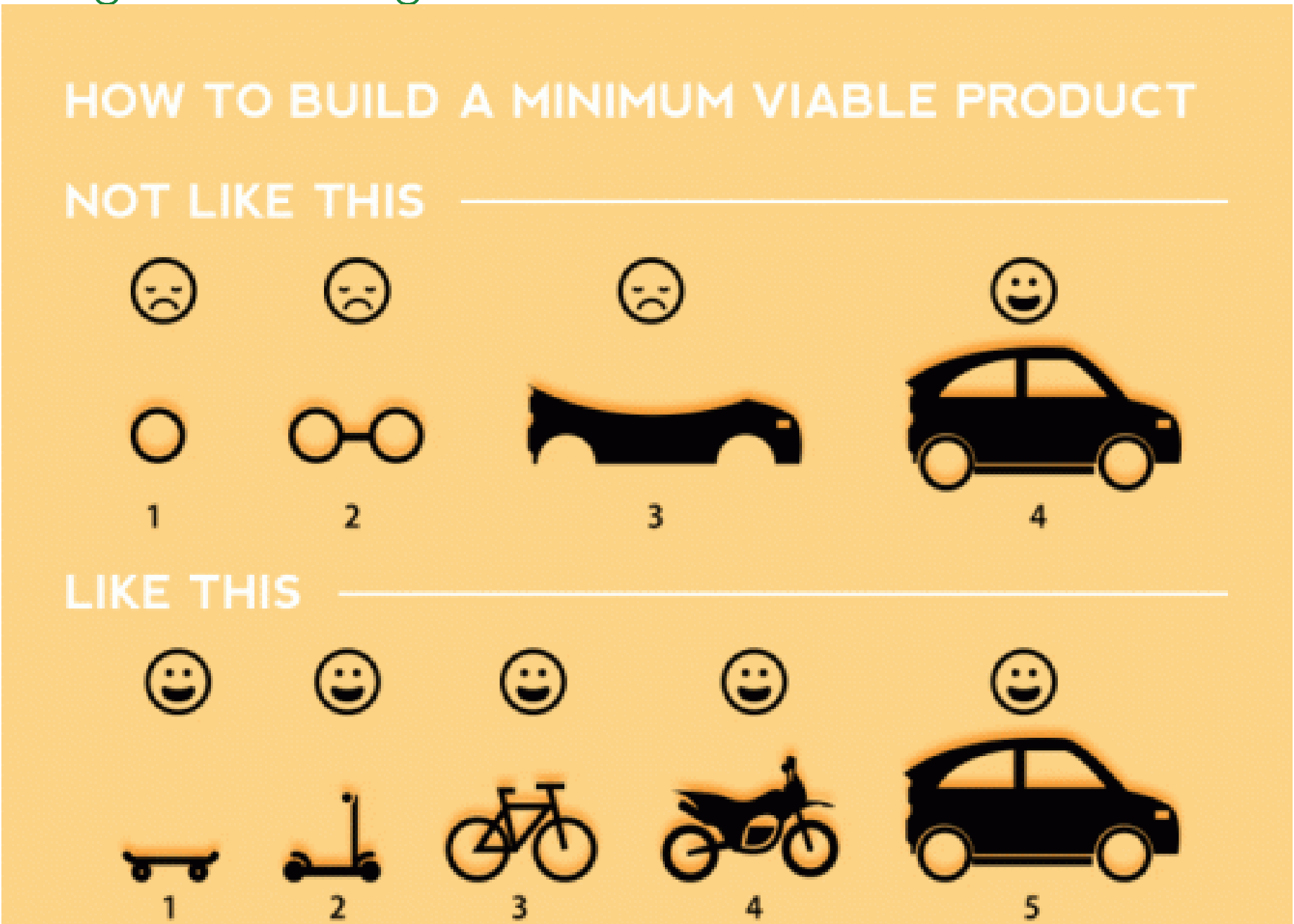


DRY vs WET

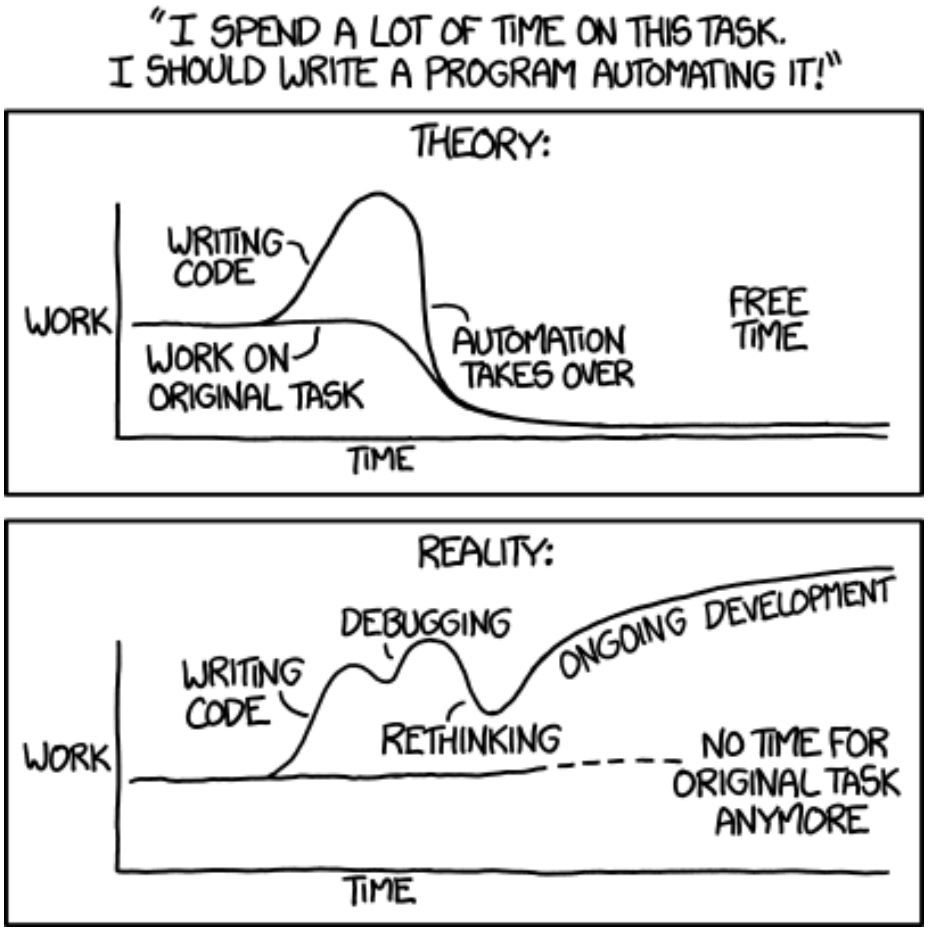
- DRY= Don't Repeat Yourself = Design Pattern
- WET= We Enjoy Typing

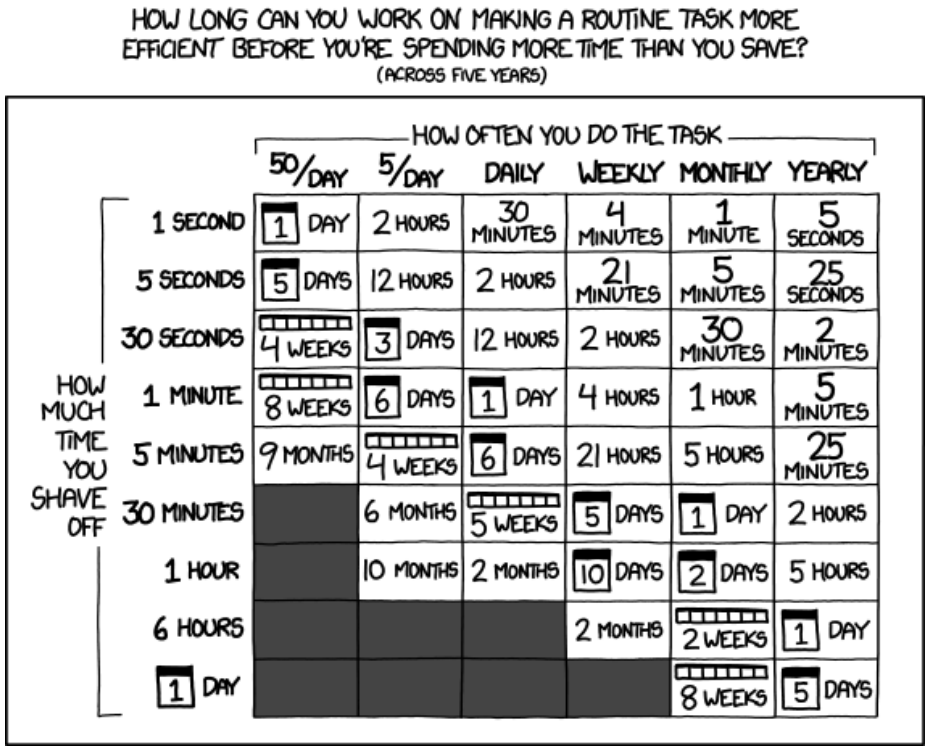
- Avoid code and data duplicates !
- ...but always keep a copy somewhere
- Target maximum redundancy: different disks, different places, different passwords, different software

Thoughts on coding



Coding process





Thank you xkcd !

Gilles Tredan Lots of Data

File Types

- Data without type = meaningless !
- Data has only value with proper decoding information.
- You need to understand (at least a bit about) data storage.
- Data format will impact:
  - Usable tools
  - Import/Export hassle (=Time!)
  - Interoperability
  - Performance

General guidelines:

- **Always** use open formats = better interoperability guarantees
- Text= slow *but* human-readable
- Binary= faster
- Some inbetween (e.g. gz dumps)

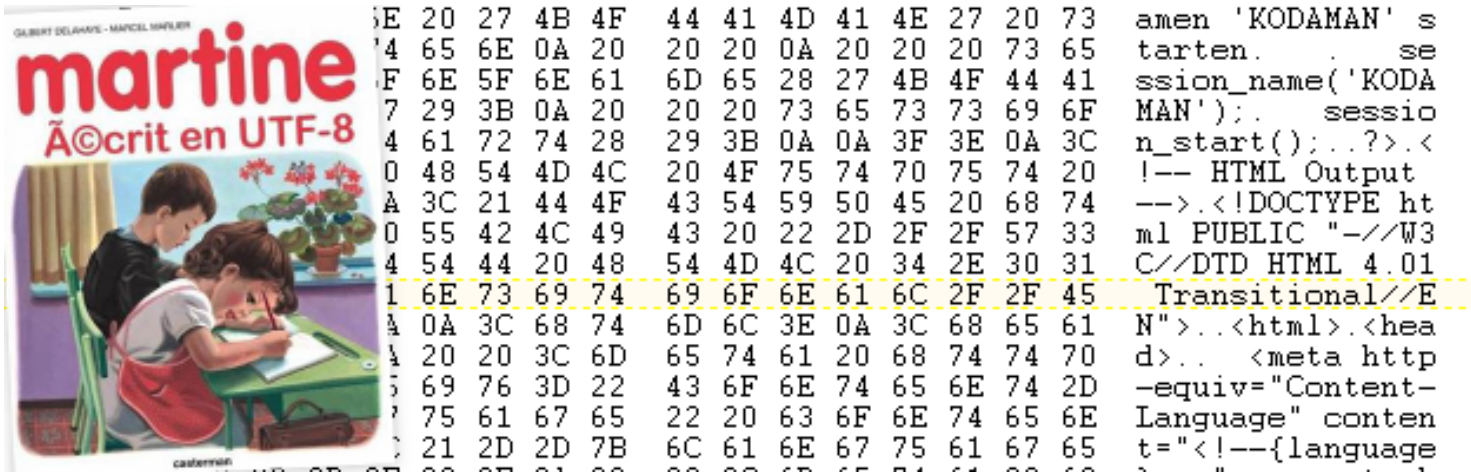
CSV

Comma separated values.

- pretty slow to parse (e.g. goto line 40)
- Very common
- I csv-ize *systematically* xls files

If size matters:

- one bit $= 1b$
- "0" $= 8b$
- "\"0\" " $= 24b$
- ""Female"" $= 64b$



Relational Databases

- Proposed by Ted Codd in the 70's
- Great reading: Serge Abitboul's inaugural lesson at Collège de France (in french..) `http://books.openedition.org/cdf/529`
- Data= a set of tables = a set of relations

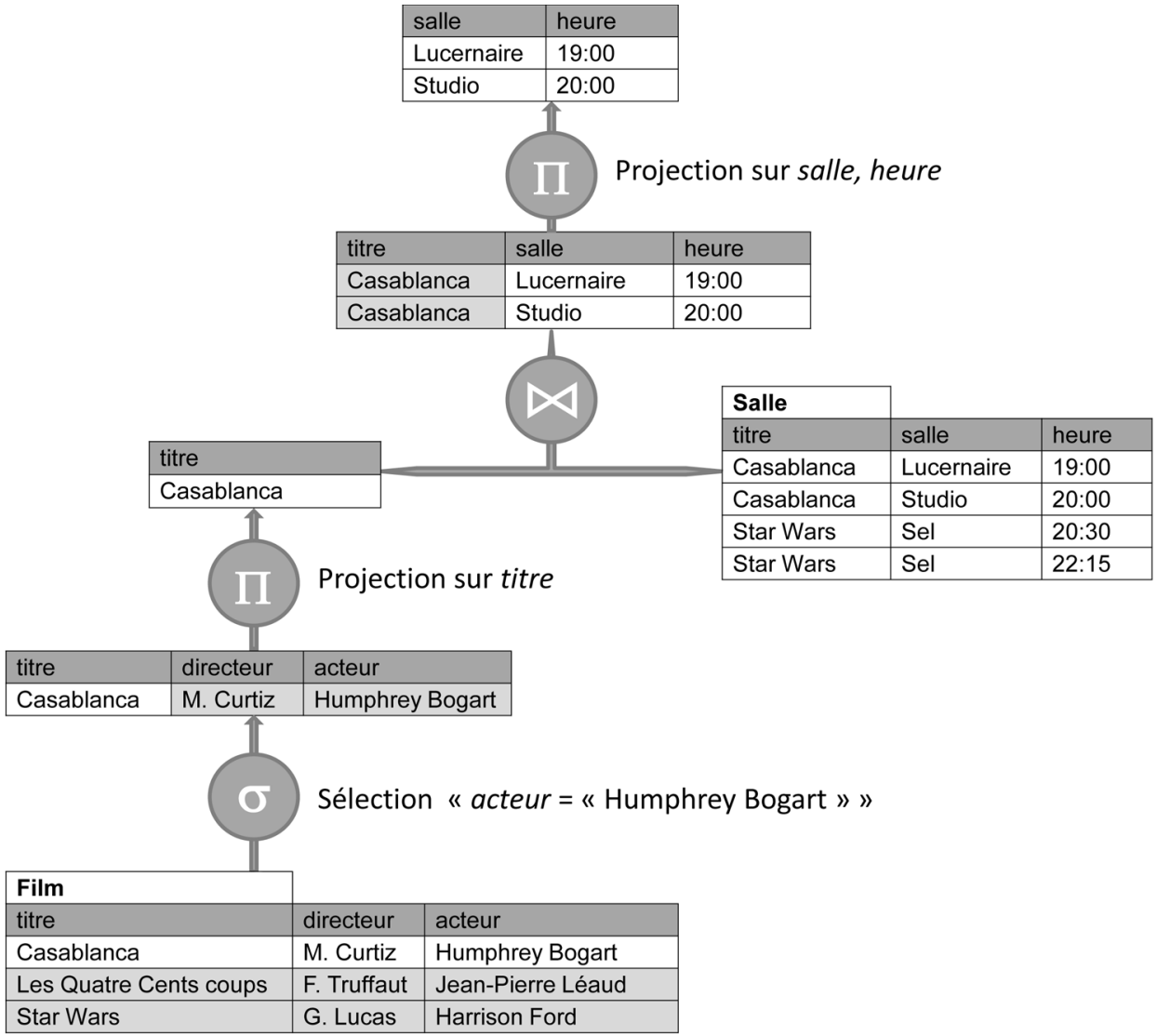
Film		
Titre	Réalisateur	Acteur
Casablanca	M. Curtiz	Humphrey Bogart
Casablanca	M. Curtiz	Peter Lorre
Les 400 coups	F. Truffaut	Jean-Pierre Léaud
Star Wars	G. Lucas	Harrison Ford

Séance		
Titre	Salle	Heure
Casablanca	Lucernaire	19:00
Casablanca	Studio	20:00
Star Wars	Sel	20:30
Stars Wars	Sel	22:15

- *(Star-wars,sel,22:15)* is a triple in the Séance relation
- A query$\simeq$a first order logic predicate:

$$\exists t, r \text{ s.t. } (Film(t, r, "HumphreyBogart") \wedge Seance(s, t, h))$$

Serge



http://books.openedition.org/cdf/529

SQL databases

- SQL = Syntax Query Langage = Langage to query RDBs
- ```
  select salle, heure
  from Film, Séance
  where Film.titre= Séance.titre and acteur= "Humphrey
  Bogart"
  ```

RDBs= dedicated to data storage and querying

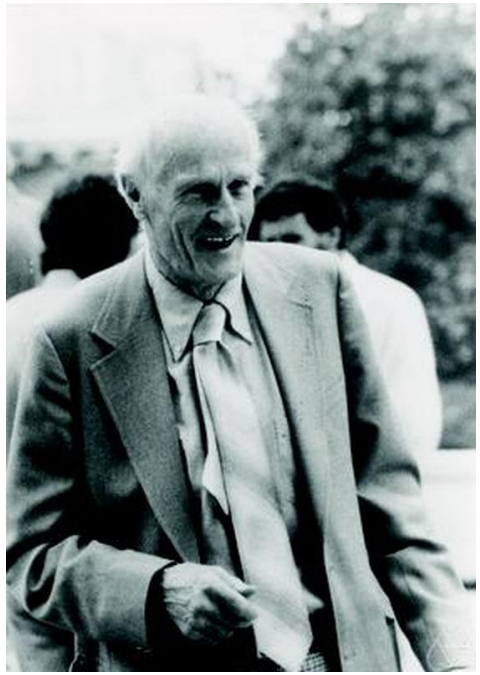
- $\Rightarrow$efficient
- Need to setup an additionnal infrastructure: complexity overhead
  ```

RegExps!

- Regexps are used to specify a set of strings
- Avoids enumeration
- Very powerfull
- Very tricky sometimes

Very useful in

- Bash
- Sql
- R
- sed, awk,...



Mr. Kleen (a *)

RegExps 2

- **Boolean "or"= | :** A vertical bar separates alternatives.
  For example, gray|grey can match "gray" or "grey".
- **Grouping= () :** Parentheses are used to define the scope and precedence of the operators (among other uses).
  gray|grey $\Leftrightarrow$ gr(a|e)y
- **Quantification** A quantifier after a a character or group specifies how often that preceding element is allowed to occur.
  - **? :** zero or one of the preceding element.
    colou?r matches both "color" and "colour".
  - *** :** zero or more of the preceding element.
    ab*c matches "ac", "abc", "abbc", "abbbc",..
  - **+ :** one or more of the preceding element.
    ab+c matches "abc", "abbc", "abbbc",.. but not "ac".
- **Position:**
  - ^ begining of a line
  - **$** end of a line

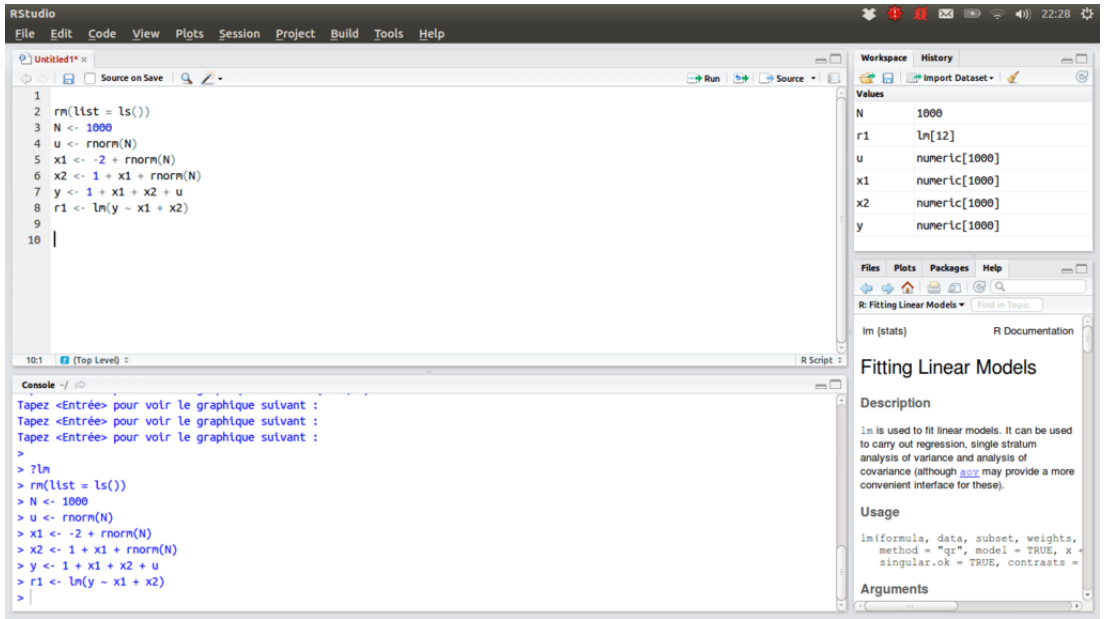
R - introduction

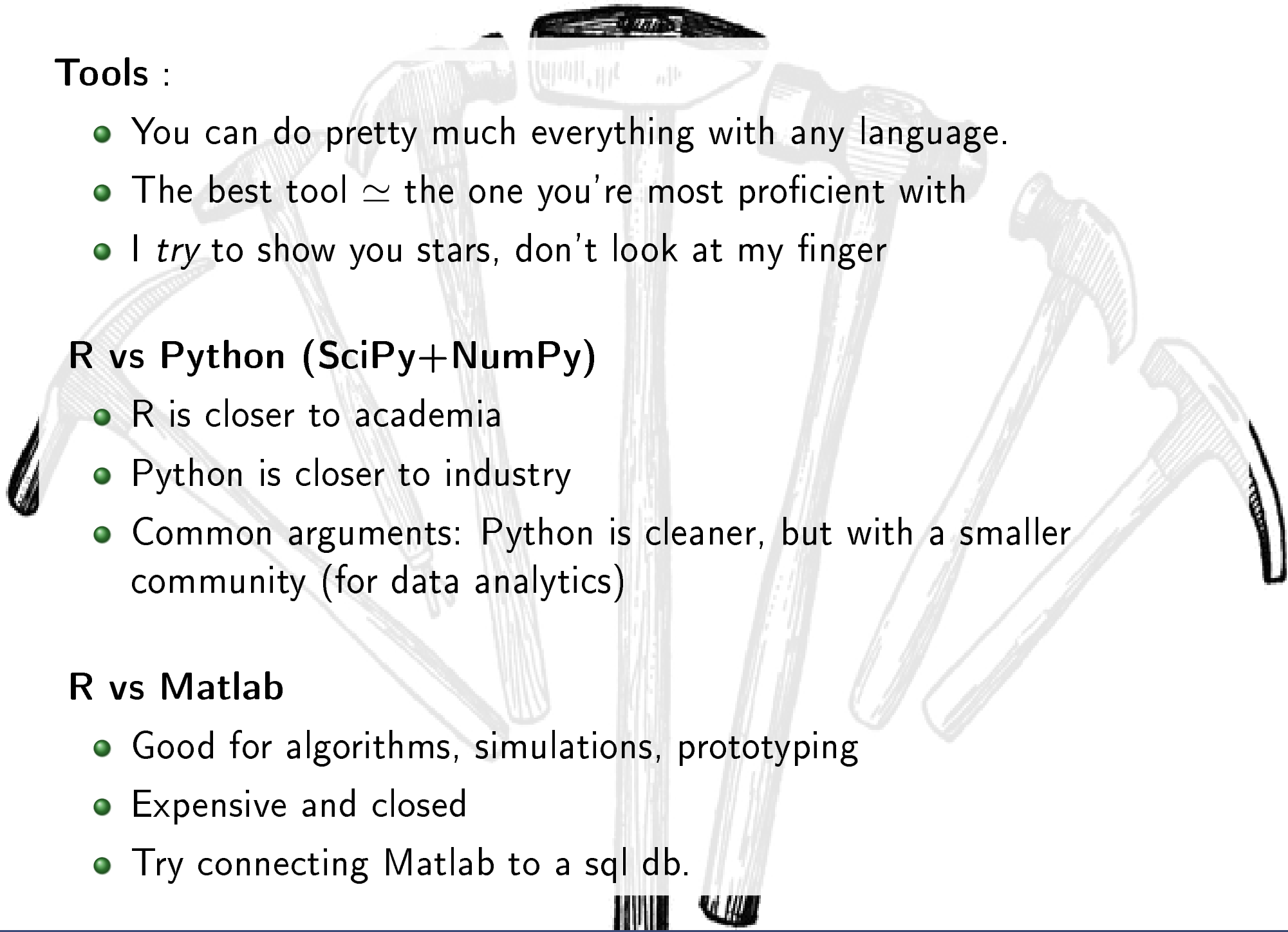
What is R ?

Interaction:

- R is an interpreted language
- Console
- Scripts
- IDE

- First release in 93
- created by Ross Ihaka and Robert Gentleman
- R = Gnu S





Tools :

- You can do pretty much everything with any language.
- The best tool $\simeq$ the one you're most proficient with
- I *try* to show you stars, don't look at my finger

R vs Python (SciPy+NumPy)

- R is closer to academia
- Python is closer to industry
- Common arguments: Python is cleaner, but with a smaller community (for data analytics)

R vs Matlab

- Good for algorithms, simulations, prototyping
- Expensive and closed
- Try connecting Matlab to a sql db.

Using R interactively

- When you use the R program it issues a prompt when it expects input commands.
- The default prompt is '>'
- Start the R program with the command `$ R`
- To quit the R program the command is `> q()` or Ctrl+d shortcut.
- At this point you will be asked whether you want to save the data from your R session.

Getting help

- Getting help with functions and features R has an inbuilt help facility similar to the man facility of UNIX.
- `> help(solve)`
- An alternative is `> ?solve`
- to `> ??solve`

R commands, case sensitivity, etc.

- R is case sensitive as are most UNIX based packages: in general $A! = a$.
- Elementary commands = expressions or assignments.
- Commands are separated either by a semi-colon (';'), or by a newline.
- Comments: starting with a hashmark ('#'), everything to the end of the line is a comment.

Typical workflow:

- Publish first paper at year 1
- Never comment code! *I know, I've written it myself..*
- Forget about the topic
- Re-open files 2 years later
- Cry
- Start commenting whatever you do

Recall and correction of previous commands

- Vertical arrow keys on the keyboard = scroll forward and backward through a command history.
- Horizontal arrow keys = moving in the selected command
- As in Bash, Ctrl+R + someText recalls the last command containing someText, Ctrl+A/E bring to the beginning/end of a line

Your best friend : Tab

- Tab= completion in console. Use it everywhere: bash R, always !
- To execute commands from a file in the working directory > `source("commands.R")`

Simple manipulations; numbers and vectors

Vectors and assignment

- R operates on named data structures.
- The simplest such structure = vector = a single entity consisting of an ordered collection
- `> x <- c(10.4, 5.6, 3.1, 6.4, 21.7)`
- This is an assignment statement using the function c()
- A number = a vector of length one.
- Notice that the assignment operator ('<-')
- In most contexts the '=' operator can be used as an alternative.
- `> c(10.4, 5.6, 3.1, 6.4, 21.7) -> x`
- `> 1/x`
- c=concatenation: `> y <- c(x, 0, x)`

Vector arithmetic

- operations on vectors are performed element by element.
- Usual elementary arithmetic operators; `+`, `-`, `*`, and `^`.
- `log`, `exp`, `sin`, `cos`, `tan`, `sqrt`, and so on, all have their usual meaning.
- `max` and `min` select the largest and smallest elements of a vector respectively.
- `range = c(min(x), max(x))`.
- `length(x)` is the number of elements in `x`.
- `sum(x)` gives the total of the elements in `x`.
- `prod(x)` their product.
- `mean(x)` and `var(x)` sample variance
- `sort(x)` (see also order() or sort.list()).
- Internally calculations are done as double precision real numbers

Generating regular sequences

- `1:30` is the vector `c(1, 2, ..., 29, 30)`.
- Warning: The colon operator has high priority: `2*1:15` is the vector `c(2, 4, ..., 28, 30)`
- Compare the sequences 1:n-1 and 1:(n-1).

Let's consider `seq()`, a more general facility for generating sequences:

- five arguments
- only some of which may be specified in any one call.
- seq(2,10) is the same vector as 2:10.
- Arguments may be named: The first two are from=value and to=value
- by=value and length=value, which specify a step size and a length for the sequence respectively
- `> seq(-5, 5, by=.2) -> s3`
- `s4 <- seq(length=51, from=-5, by=.2)`

Logical vectors

- The elements of a logical vector can have the values TRUE, FALSE, and NA.
- The first two are often abbreviated as T and F
- Logical vectors are generated by conditions: `> temp <- x > 13`

Logical operators:

- The logical operators are $<, <=, >, >=, ==$ for exact equality and != for inequality.
- $c1 \& c2$= intersection ("and"), $c1 \| c2$ = union ("or"), and $!c1$ is the negation of c1.
- Logical vectors may be coerced into numeric vectors: FALSE=0 and TRUE=1.

Missing Values

- Components of a vector may not be completely known.
- $\Rightarrow$assigned with the special value NA.
- In general any operation on an NA becomes an NA.
- `is.na(x)[i]==T` iff `x[i]==NA`
- `> z <- c(1:3,NA); ind <- is.na(z)`
- Warning: x == NA is not is.na(x)
- Second kind of "missing" values : Not a Number, NaN, values.
- `> 0/0` , `> Inf - Inf`
- is.na(xx) is TRUE both for NA and NaN values.

Character vectors

- denoted by a sequence of characters delimited by the double quote character, e.g., "x-values", "New iteration results".
- Character strings are entered using either matching double (") or single (') quotes
- They use C-style escape sequences, using \ as the *escape character* inside double quotes " is entered as \".
- Character vectors are concatenated into a vector by `c()`
- `paste()` function takes an arbitrary number of arguments and concatenates them one by one into character strings.
- arguments are by default separated in the result by a single blank character

```
> labs <- paste(c("X","Y"), 1:10, sep="")
```

Index vectors; selecting and modifying subsets of a data set

- Subsets of the elements of a vector may be selected using an index vector
- Very important ! Also for arrays, data frames, lists, etc.
- can also appear on the receiving end of an assignment,
  - `> x[is.na(x)] <- 0`
  - `> y[y < 0] <- -y[y < 0]`
  - `> y <- abs(y)`
- Such index vectors can be any of four distinct types.

Index Vectors

1. A logical vector. Values corresponding to TRUE in the index vector are selected and those corresponding to FALSE are omitted.
   ```
   > y <- x[!is.na(x)]
   > (x+1)[(!is.na(x)) & x>0] -> z
   ```
2. A vector of positive integers. The corresponding elements of the vector are selected and concatenated, in that order
   ```
   > x[1:10]
   ```
3. A vector of negative integers. specifies the values to be excluded
   ```
   > y <- x[-(1:5)]
   ```
4. A vector of character strings.
   only applies where an object has a names attribute to identify its components
   - ```
     > fruit <- c(5, 10, 1, 20)
     > names(fruit) <- c("orange", "banana", "apple", "peach")
     > lunch <- fruit[c("apple","orange")]
     ```
   - particularly useful with data frames
     ```

Which

`which` allows to pass from a logical index vector to an integer index vectors.

- `> which(c(T,F,T,F,F,F,T))`
  `1 3 7`
- They are *semantically* strictly equivalent
  `> start`
  `"a" "b" "c" "d" "e" "f" "g"`
  `> start`
  $c(T, F, T, F, F, F, T)$
  `"a" "c" "g"`
  `> start`
  $which(c(T, F, T, F, F, F, T))$
  `"a" "c" "g"`
- *You* prefer index vectors, ok
- It is more expensive !

More complex objects

About objects

Vectors are the most important type of object in R

but there are several others

- matrices or more generally arrays = multi-dimensional generalizations of vectors
- factors = pain
- lists are a general form of vector
- data frames = 'data matrices' with one row per observation.
- functions are themselves objects in R

Vectors must have their values all of the same mode : **atomic**

lists are known as "recursive" rather than atomic structures

About objects

Objects have **attributes**

- length: `length(object)`

```
> e <- numeric()
> e[3] <- 17
> length(alpha) <- 3
```

- class:

```
> z <- 0:9
> digits <- as.character(z)
> d <- as.integer(digits)
```

large collection of functions of the form as.something()

- An "empty" object may still have a class. For example
- All objects in R have a class, reported by the function `class(object)`
- Simple vectors: "numeric", "logical", "character" or "list"

Factors

A factor = discrete classification vector (grouping)

- `set.seed(124)`
  `schtyp <- sample(0:1, 20, replace = TRUE)`
  `is.factor(schtyp)`
  `is.numeric(schtyp)`
- Now let's create a factor variable called schtyp.f.
- private, will correspond to schtyp=0
- public, will correspond to schtyp=1
- the order of the labels will follow the numeric order of the data.
- `schtyp.f <- factor(schtyp, labels = c("private", "public"))`
  `is.factor(schtyp.f)`

Factors(2)

- Let's generate a string variable called ses (socio-economic status).

```
ses <- c("low", "middle", "low", "low", "low", "low",
"middle", "low", "middle", "middle", "middle",
"middle", "middle", "high", "high", "low", "middle",
"middle", "low", "high")
ses.f.bad.order <- factor(ses)
levels(ses.f.bad.order)
```

- Problem: the levels are ordered according to the alphabetical order
- Thus, "high" is the lowest level of ses.f.bad.order
- we need to use the levels argument to indicate the correct ordering of the categories.

```
ses.f <- factor(ses, levels = c("low", "middle",
"high"))
levels(ses.f)
```

Arrays

- *Dimension vector* = a vector of non-negative integers.
- length= k $\Rightarrow$the array is k-dimensional,
- a matrix is a 2-dimensional array.
- A vector = an array with a dimension vector as its dim attribute.

```
> z <- rep(1,1500)
> dim(z) <- c(3,5,100)
```

Array indexing. Subsections of an array

- Subsections of an array may be specified by giving a sequence of index vectors v
- if any index position is given an empty index vector, then the full range of that subscript is taken.
- `> c(a[2,1,1], a[2,2,1], a[2,3,1], a[2,4,1], a[2,1,2], a[2,2,2], a[2,3,2], a[2,4,2])`
- A matrix may be used with a single index matrix

```
> x <- array(1:20, dim=c(4,5))   # Generate a 4 by 5 array.
> x
     [,1] [,2] [,3] [,4] [,5]
[1,]    1    5    9   13   17
[2,]    2    6   10   14   18
[3,]    3    7   11   15   19
[4,]    4    8   12   16   20
> i <- array(c(1:3,3:1), dim=c(3,2))
> i                          # i is a 3 by 2 index array.
     [,1] [,2]
[1,]    1    3
[2,]    2    2
[3,]    3    1
> x[i]                       # Extract those elements
[1] 9 6 3
> x[i] <- 0                  # Replace those elements by zeros.
> x
     [,1] [,2] [,3] [,4] [,5]
[1,]    1    5    0   13   17
[2,]    2    0   10   14   18
[3,]    0    7   11   15   19
[4,]    4    8   12   16   20
```

Matrix facilities

- a matrix is just an array with two subscripts
- R contains many operators and functions that are available only for matrices.
- For example `t(X)`, `nrow(A)` and `ncol(A)`
- Multiplication : %*% is used for matrix multiplication.
- If A and B are square matrices of the same size, then
  > `A * B` is the matrix of element by element products
  > `A %*% B` is the matrix product.
- diag(v), where v is a vector, gives a diagonal matrix v as the diagonal.

Solving linear equations is the inverse of matrix multiplication

- `> b <- A %*% x`
- If only A and b are given, the vector x is the solution of that linear equation system.
- `> solve(A,b)` solves the system, returning x (up to some accuracy loss).
- inverse of A: `solve(A)`
- Numerically, it is both inefficient and potentially unstable to compute x <- solve(A) %*% b instead of solve(A,b).

Eigenvalues and eigenvectors

- `eigen(Sm)` calculates the eigenvalues and eigenvectors of a symmetric matrix Sm.
- result= a list of two components named values and vectors.
- `> ev <- eigen(Sm)` will assign this list to ev.
- `ev$val` = the vector of eigenvalues of Sm
- `ev$vec` = the matrix of corresponding eigenvectors.
- Had we only needed the eigenvalues

```
> evals <- eigen(Sm)$values
```

- For large matrices if eigenvectors are not needed use the expression

```
> evals <- eigen(Sm, only.values = TRUE)$values
```

Lists and Dataframes

Lists

- list = an ordered collection of heterogeneous objects
- `> Lst <- list(name="Fred", wife="Mary", no.children=3, child.ages=c(4,7,9))`
- Components are always numbered: `Lst[[1]]`, `Lst[[2]]`,...
- `length(Lst)` gives the number of (top level) components
- Components of lists may also be named
  `> name$component_name`
- `Lst$name` is the same as `Lst[[1]]` and is the string "Fred",
- `Lst$wife` is the same as `Lst[[2]]` and is the string "Mary",
- `Lst[["name"]]` is the same as `Lst$name`.
  `> x <- "name"; Lst[[x]]`
- It is very important to distinguish `Lst[[1]]` from `Lst[1]`
- `'[[...]]'` is the operator used to select a single element
- `'[...]'` is a general subscripting operator.

Data Frames

- A data frame is a list with class "data.frame".
- The components must be vectors (numeric, character, or logical), factors.
- Vector structures appearing must all have the same length,
- By default character vectors are coerced to be factors.

Short: data frame = one observation = a matrix with columns possibly of differing modes and attributes.

- `> accountants <- data.frame(home=statef, loot=incomes, shot=incomef)`
- use the read.table()

Data Frame Manipulation

Splitting

- by hand: `diamonds[,c("cut","color","clarity")]` `diamonds[1:100,]`
- using a function: `split(mtcars,mtcars$gear)`
- many other examples ahead

Pasting

- `cbind`: to combine the columns of two datasets
- `rbind`: to combine the rows of two datasets

Ordering

- `arrange(df,value)`
- `select,subset`

Summary about types

1D	Vector	List
2D	Matrix	Data Frame
n Dimensionnal	Array	Lists of lists
Content	Homogeneous	Heterogeneous

- If you can fluidly transform your data between any of these formats, R will be your best friend

Statistics made short

some useful functions

Observation

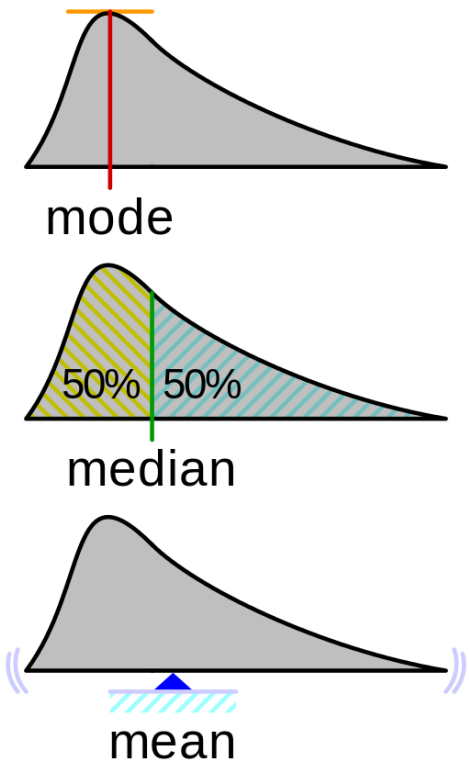
- `sample`
- `head`
- `tail`

Location

- `mean`
- `median quantiles, min/max`
- `summary`

Dispersion

- Range
- Standard deviantion (`sd`)
- ecdf



Example EDA

Porto weather data, over year 2014

- l.temp, h.temp, ave.temp : lowest, highest and average temperature for the day (in C)
- l.temp.time, h.temp.time : hour of the day when l.temp and h.temp occurred rain : amount of precipitation (in mm)
- ave.wind : average wind speed for the day (in km/h)
- gust.wind : maximum wind speed for the day (in km/h)
- gust.wind.time : hour of the day when gust.wind occurred dir.wind : dominant wind direction for the day

Thank you Pedro M., Analytical Minds

Some functions to think about

Similarity

- Jaccard index $J(A,B) = \frac{|A \cap B|}{|A \cup B|} = \frac{|A \cap B|}{|A| + |B| - |A \cap B|}$.
- Cosine similarity similarity $= \cos(\theta) = \frac{\mathbf{A} \cdot \mathbf{B}}{\|\mathbf{A}\| \|\mathbf{B}\|} = \frac{\sum_{i=1}^{n} A_i B_i}{\sqrt{\sum_{i=1}^{n} A_i^2} \sqrt{\sum_{i=1}^{n} B_i^2}}$
- Euclidean distance

Correlations/ranks

- Pearson
- Spearman ρ
- Kendall τ

```
cor(x, y, method = c("pearson", "kendall", "spearman"))
```

Distribution of values

- hist(x)
- Empirical Cumulative Distribution Function `ecdf(x)`
- Like histogram, avoids bining problem

Probability Distributions

Distribution	R name	additional arguments
beta	beta	shape1, shape2, ncp
binomial	binom	size, prob
chi-squared	chisq	df, ncp
exponential	exp	rate
gamma	gamma	shape, scale
geometric	geom	prob
hypergeometric	hyper	m, n, k
normal	norm	mean, sd
Poisson	pois	lambda
Student's t	t	df, ncp
uniform	unif	min, max
Weibull	weibull	shape, scale

- `p$name` gives cdf $P(X \leq x)$,
- `d$name` gives the pdf
- `q$name` gives quantile function
  given q, the smallest x such that $P(X \leq x) > q$

Writing Functions

Flow Control

Commands may be grouped together in braces, `{expr1; ...; exprM}`, the value of the group = the result of the last expression in the group evaluated.

Conditional execution: if statements

- `> if (expr1) expr2 else expr3`
- && and ||!= & and |
- || apply to vectors of length one, and only evaluate their second argument if necessary
- the ifelse function = Vectorized version of the if
- `c<-ifelse(condition, a, b)`: $\forall i$ $a[i] = c[i]$ if $condition[i]$, and $b[i]$ otherwise.

Flow Cont.

- `> for (name in expr1) expr2`
- name is the loop variable.
- expr1 is a vector expression, (often a sequence like 1:20)
- Warning: `for()` loops are used in R code much less often than in compiled languages.
- Code that takes a 'whole object' view is likely to be both clearer and faster in R.
- `> while (condition) expr`
- The `break` statement can be used to terminate any loop, possibly abnormally.
- The `next` statement can be used to discontinue one particular cycle and skip to the "next".
- Of course break and next are dirty too...

Functions

Learning to write useful functions is **the** way to make your use of R comfortable and productive.

A *function* is defined by an assignment of the form `> name <- function(arg1, arg2, ...) expression`

- Named arguments:
- ```
  > fun1 <- function(data, data.frame, graph, limit) {
  [function body omitted] }
   > ans <- fun1(d, df, TRUE, 20)
   > ans <- fun1(d, df, graph=TRUE, limit=20)
   > ans <- fun1(data=d, limit=20, graph=TRUE,
  data.frame=df)
  ```
- are all equivalent.

Explicit **return**: `return(value)`. Implicit return: result of the last command of the block
  ```

Functions(2)

Default values:

- `> fun1 <- function(data, data.frame, graph=TRUE, limit=20) { ... }`
  could be called as `> ans <- fun1(d, df)`
- `> ans <- fun1(d, df, limit=10)` which changes one of the defaults.

The '...' argument

- ```
  fun1 <- function(data, data.frame, graph=TRUE, limit=20,
  [omitted statements]
  if (graph)
    par(pch="*", ...)
  [more omissions]
  }
  ```

Assignments within functions: assignments done within the function are local and temporary and are lost after exit from the function.
  ```

Some useful functions

Vectorized functions

Like in Matlab, R heavily relies on vectorized functions.

Apply-like functions= Functionnals

- `lapply(list l,function f,...)`: Applies f to each element of l: return $list(f(l[1],...), f(l[2],...),...)$
- `apply(array a,margin,function f,...`: Similar to lapply, but for arrays. `margin=1` applies f to each *row*, `margin=2` applies f to each `column`.
- Many other variations (`rollapply`,`sapply`,`vapply`).

Vectorized versions

- Many functions have a vectorized version
- Check `cumSum` ($res[i] = \sum_{j=1}^{i} input[i]$,`rle` (run length encoding).

Always prefer the Vectorized version !

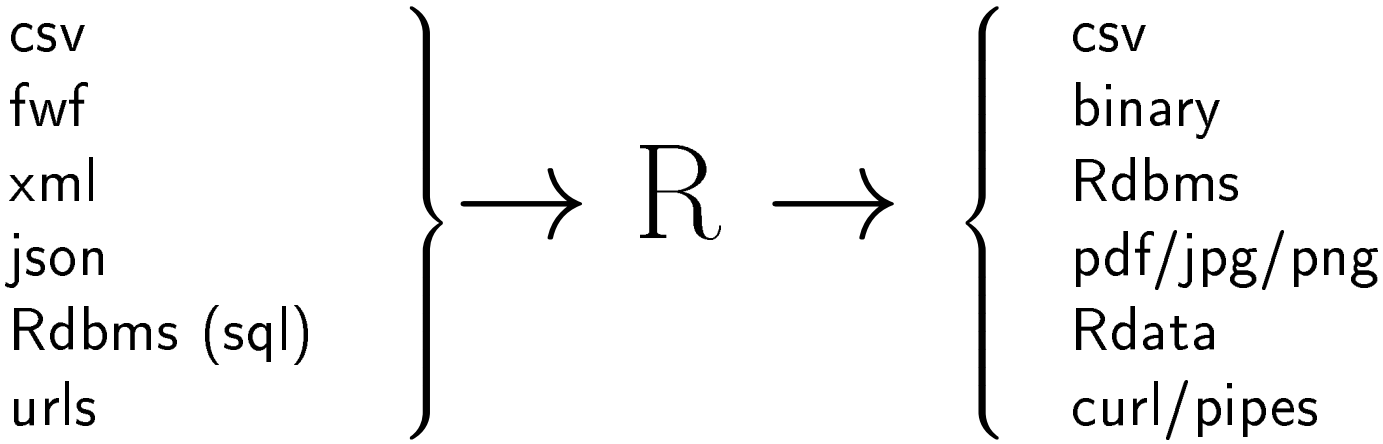
Transforming data

Some very usefull functions:

- `which, which.min, which.max`
- `do.call(cbind,list)` (similar to xargs)
- `do.call(c,as.list(letters))`
- in general, drop a column/row: `df$column<-NULL`

Connecting R to the remaining world

R is just an element of the pipeline



- Pain = encodings
- unix integration
- compression: gz and zip

Central question: Where shall I implement this processing ?

- Should I search files in bash, plot with R, and compress results with Tar?
- Should I do everything in R ?

R common file operations

- Like all processes, R has a working directory
- `getwd()`,`setwd()`
- ```
  cat("file A\n", file = "A")
  cat("file B\n", file = "B")
  file.append("A", "B")
  file.create("A")
  file.append("A", rep("B", 10))
  if(interactive()) file.show("A")
  file.copy("A", "C")
  dir.create("tmp")
  file.copy(c("A", "B"), "tmp")
  list.files("tmp")
  ```
  ```

Importing csv

- 3 functions namely:`read.table`, `read.csv`, `read.csv2` (adapted to european conventions)
- Default parameters to watch
  - `header=TRUE`
  - `sep = ","`
  - `comment.char = ""`
  - `stringsAsFactors=T`
- *Note:* you can also read urls: `read.table("http://www.ats.ucla.edu/stat/data/test.txt", header = TRUE))`

- `flist <- list.files(resdir,pattern="*pdf",full.names=T)`
- `datalist=lapply(flist,read.table)`
- `lapply(datalist,process)`

Exporting data

In **CSV**

- `write.csv(df,filename,...)`
- Only for data frames
- Csv supported everywhere
- You lose: factor levels
- Compression is easy:
  `write.csv(myDF, file = bzfile("myDF.csv.bz2"))`

As R objects

- `save(objects,file=...)`, `saveRDS(object,file=...)`
- keeps everything in place
- Good for checkpointing
- `load(file=...)`, `obj<-readRDS(file=...)`

Typical workflow

Directory organisation

- `load.R` : data import functions. possibly ended by a `save()`
- `clean.R` : dirt cleaning
- `func.R` : all functions definitions
  no processing inside $\Rightarrow$can be sourced for free
- `do.R` : all actual processing (*i.e.* where functions are called)

Working with collegues

- Agree on a format: files names **and** file contents
- Insist to get column names
- Additionnal columns get at the end !
- Protest when not respected !

R: Batteries included !

R package system

- R has an integrated package system
- to install a package, type `install.packages("name")`
- select a mirror to download the package from
- `library(name)` to load the package: that's it !
- 6779 packages installed on cran
- there is *whp* a package solving your problem
- and if not `devtools`

Note: for packages requiring a tighter system integration (e.g. external libraries), try apt-get first.

Some useful R packages

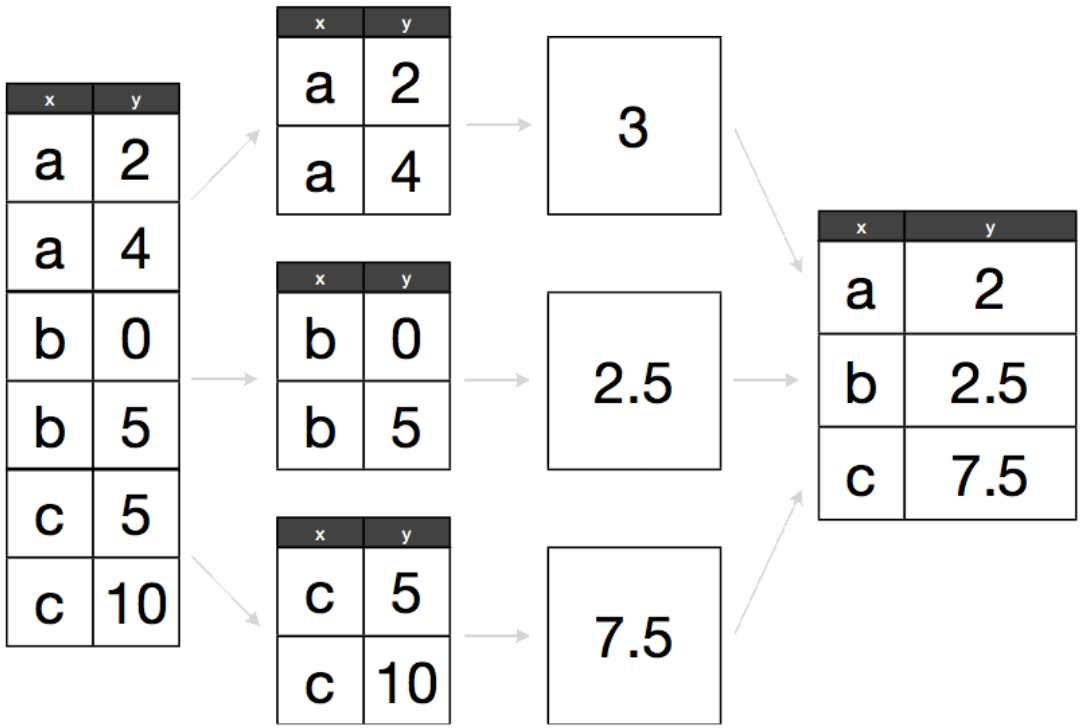
igraph	Network analysis and visualization
doMC	Foreach parallel
snow	Simple Network of Workstations
ggplot2	An implementation of the Grammar of Graphics
(d)plyr	Tools for splitting, applying and combining data
reshape2	Flexibly reshape data: a reboot of the reshape package.
rjson	JSON for R
RMySQL	R interface to the MySQL database
stringr	Make it easier to work with strings.
lubridate	Make it easier to work with dates.
data.table	Manipulate big data frames
zoo	Time Series (Z's ordered observations)

Data manipulation

Plyr

One good strategy to handle loads of data:

- Split: a problem into manageable (meaningful) pieces
- Apply: process each piece independently
- Combine: the obtained results together!



Hadley Wickham, author of ggplot, reshape, dplyr, plyr !

Plyr in practice

plyr provides a family of functions of the form:

- `a*ply(.data, .margins, .fun, ..., .progress = "none")`
- `d*ply(.data, .variables, .fun, ..., .progress = "none")`
- `l*ply(.data, .fun, ..., .progress = "none")`

replace the * depending on the output:

- `a` for an array
- `d` for a data frame
- `l` for a list
- `_` to discard the output

Ddply is your friend

- Transform: add fields computed relatively to categories
  `ddply(msleep, vore,transform, rank=rank(-sleep_total))`
- Summarize: group by category, and return aggregate stats
  `ddply(msleep, vore,summarize, med=median(sleep_total)))`
- In general: provide a complete function, process by group, and return a data frame.

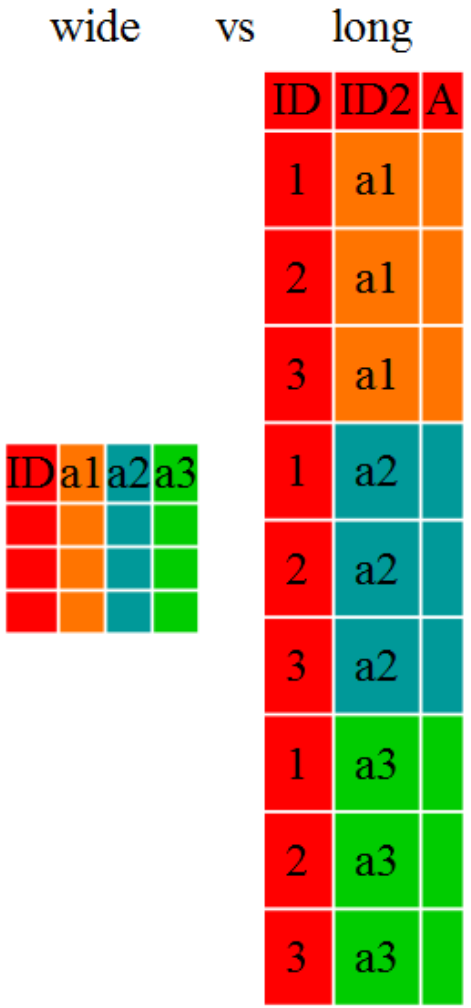
Reshape

melt

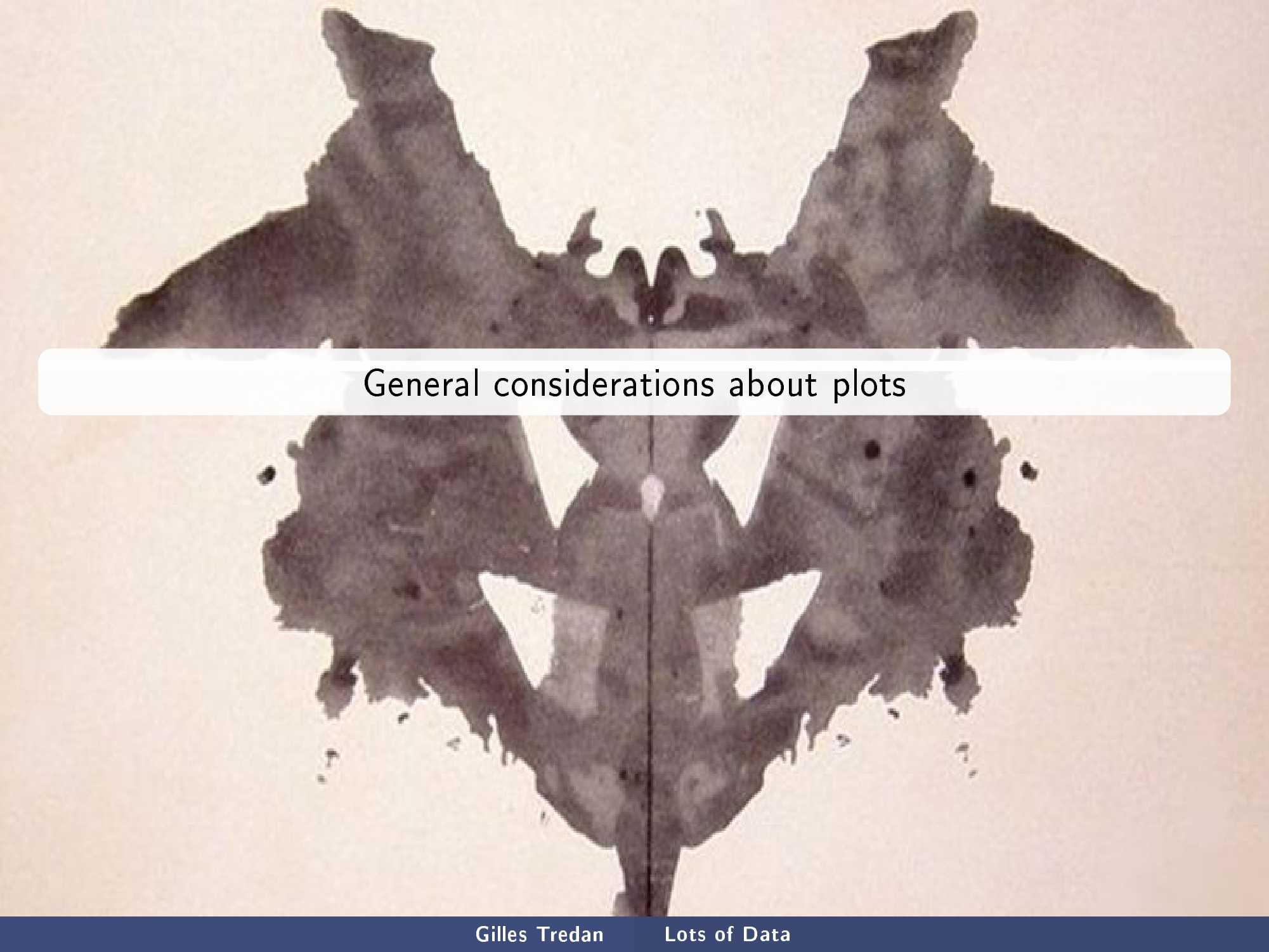
- Transform a large data frame in a long one
- Best friend of ggplot
- `id.vars` will not be melted. **measure.vars** will be melted

cast

- Opposite of melt
- Somewhat less useful, but still good to know
- Especially useful for producing tables



(Gg)plot



General considerations about plots

Carte Figurative des pertes successives en hommes de l'Armée Française dans la campagne de Russie 1812-1813.

Dressée par M. Minard, Inspecteur Général des Ponts et Chaussées en retraite.

Paris, le 20 Novembre 1869.

Les nombres d'hommes présents sont représentés par les largeurs des zônes colorées à raison d'un millimètre pour dix mille hommes; ils sont de plus écrits en travers des zônes. Le rouge désigne les hommes qui entrent en Russie, le noir ceux qui en sortent. _____ Les renseignements qui ont servi à dresser la carte ont été puisés dans les ouvrages de M.M. Chiers, de Ségur, de Fezensac, de Chambray et le journal inédit de Jacob, pharmacien de l'Armée depuis le 28 Octobre.

Pour mieux faire juger à l'oeil la diminution de l'armée, j'ai supposé que les corps du Prince Jérôme et du Maréchal Davoust qui avaient été détachés sur Minsk et Mohilow et ont rejoint vers Orscha et Witebsk, avaient toujours marché avec l'armée.

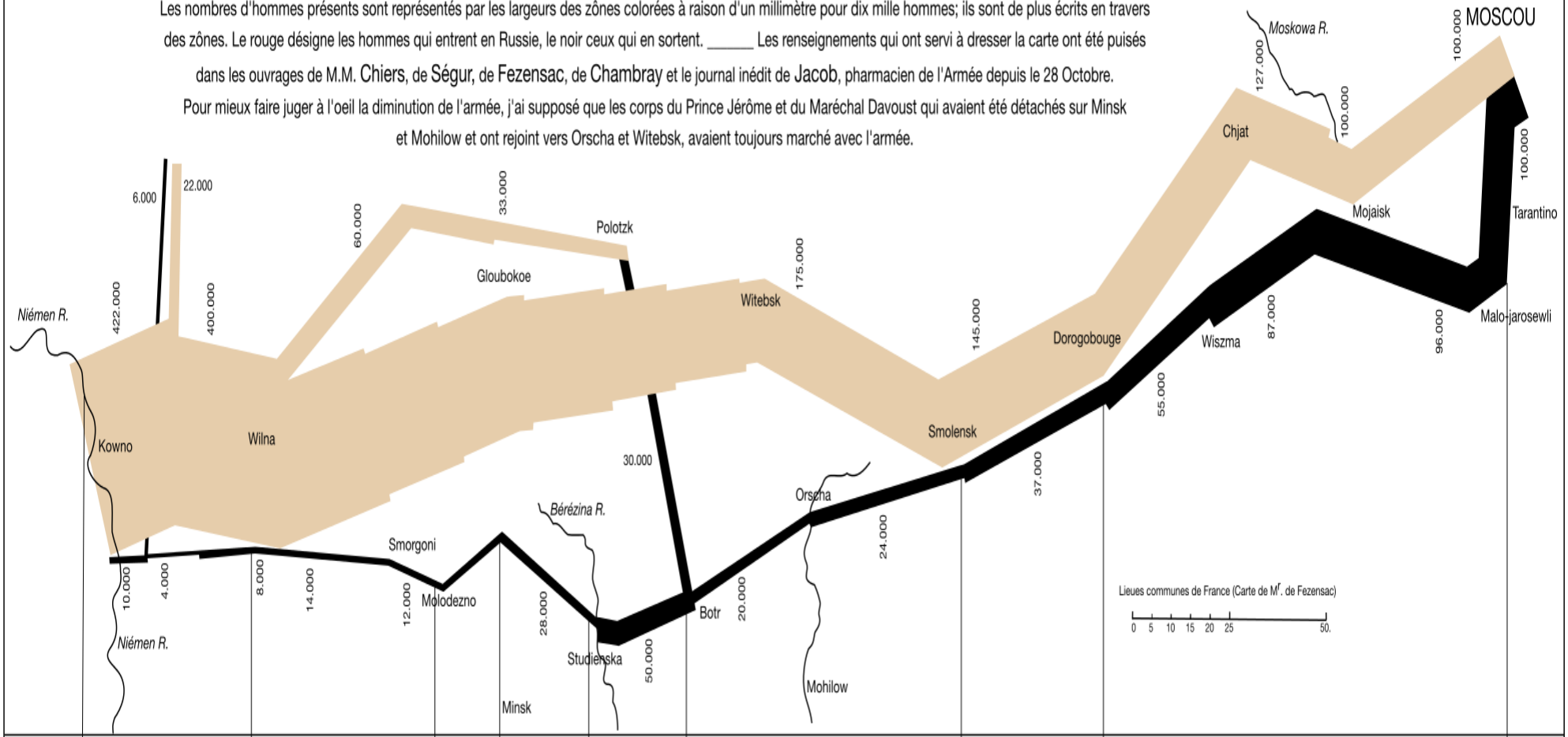
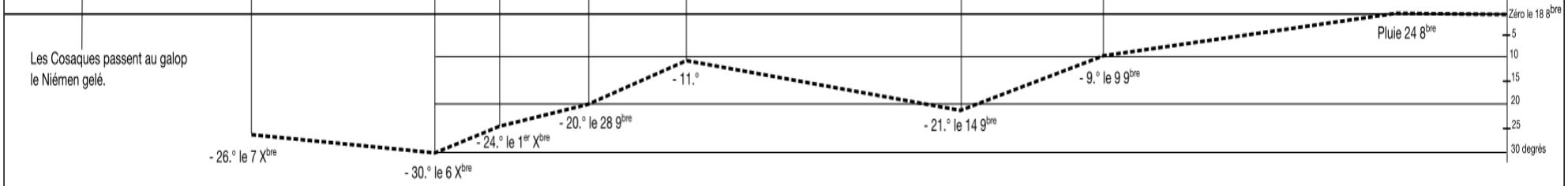
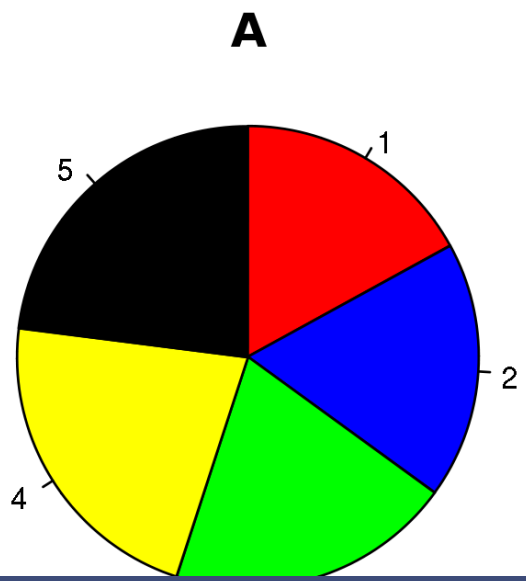
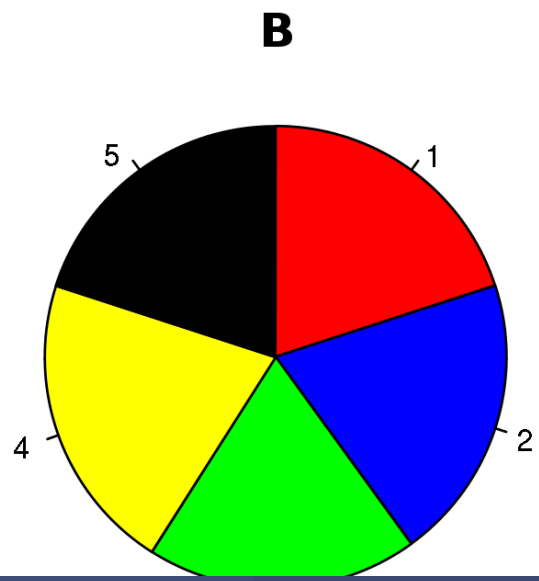


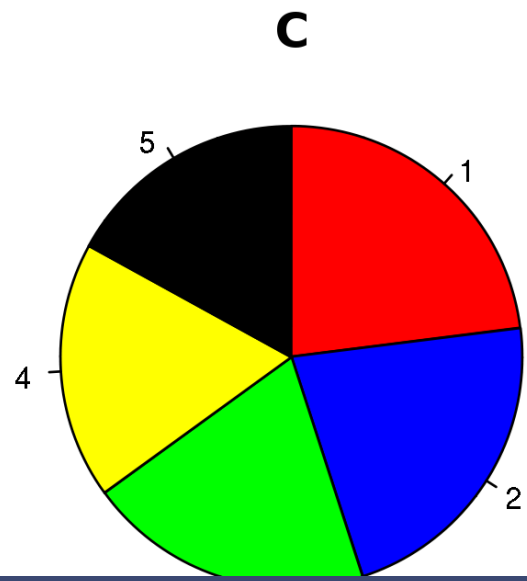
TABLEAU GRAPHIQUE de la température en degrés du thermomètre de Réaumur au dessous de zéro.



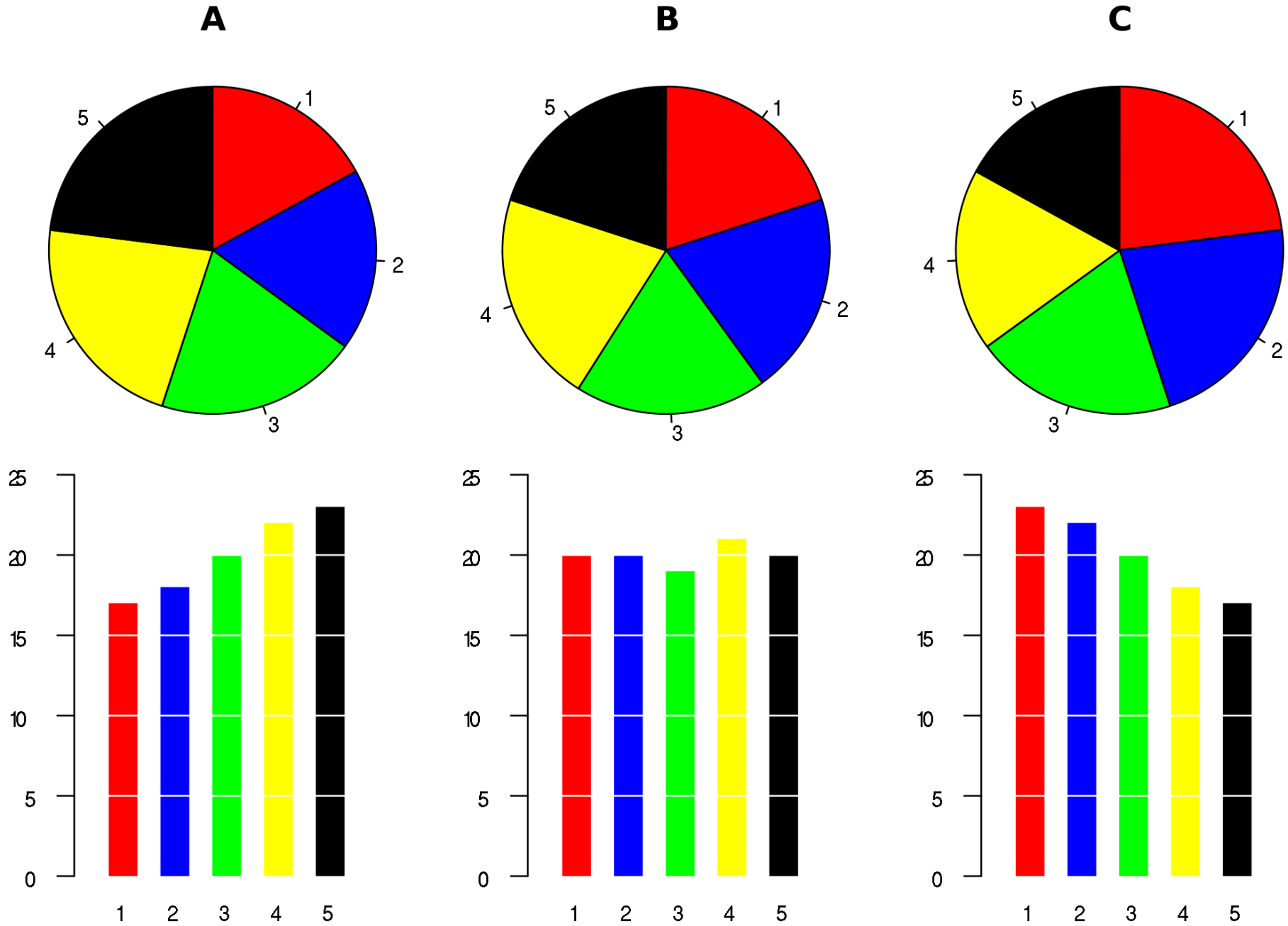
Pie charts are bad !







Pie charts are bad !



Which plot for wich content ?

- Usually, factors call for histograms
- `reorder` is your friend
- Note: boxplots are not standardized, they need explanation!

- budget $\times$ number of votes: binning problem
- Ecdfs avoid this problem !
- Get used to reading them, but be sure your audience can do it too !
- Get correct labels on the axis

- Protip: use tables if you wanna flood your audience with data !

Grammar of Graphics

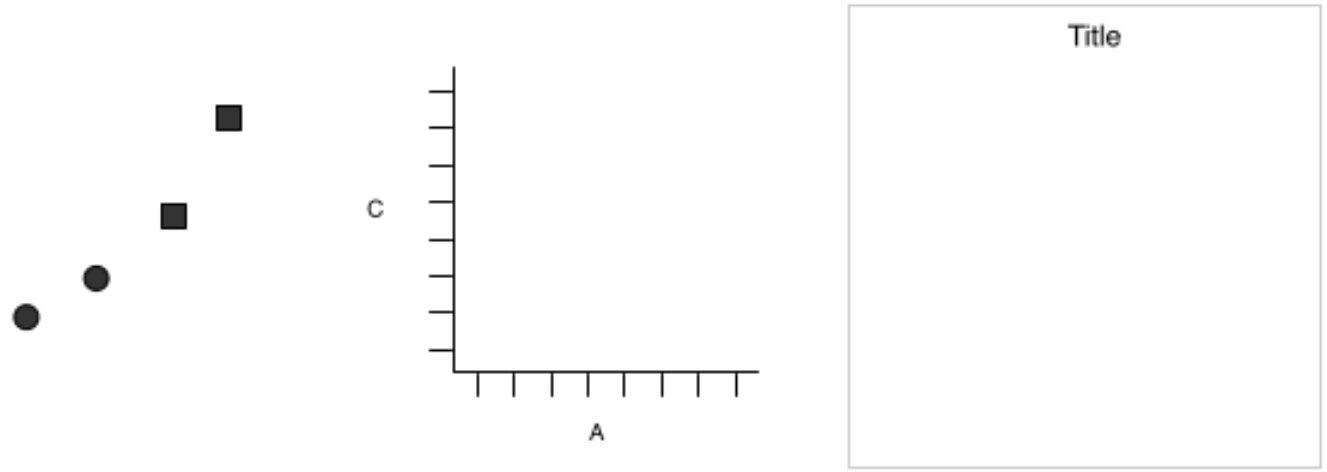


Figure 1. Graphics objects produced by (from left to right): geometric objects, scales and coordinate system, plot annotations.

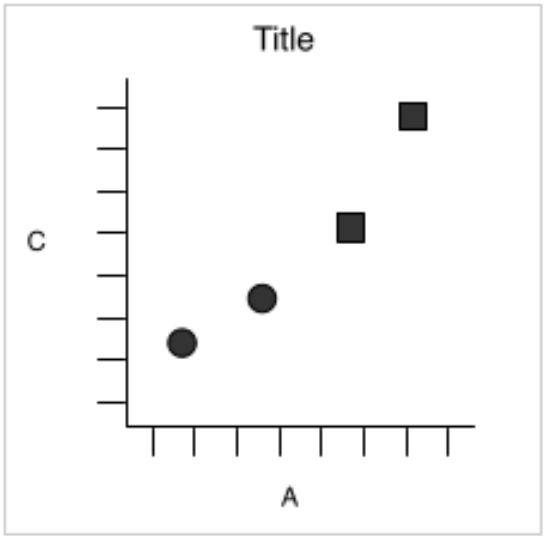
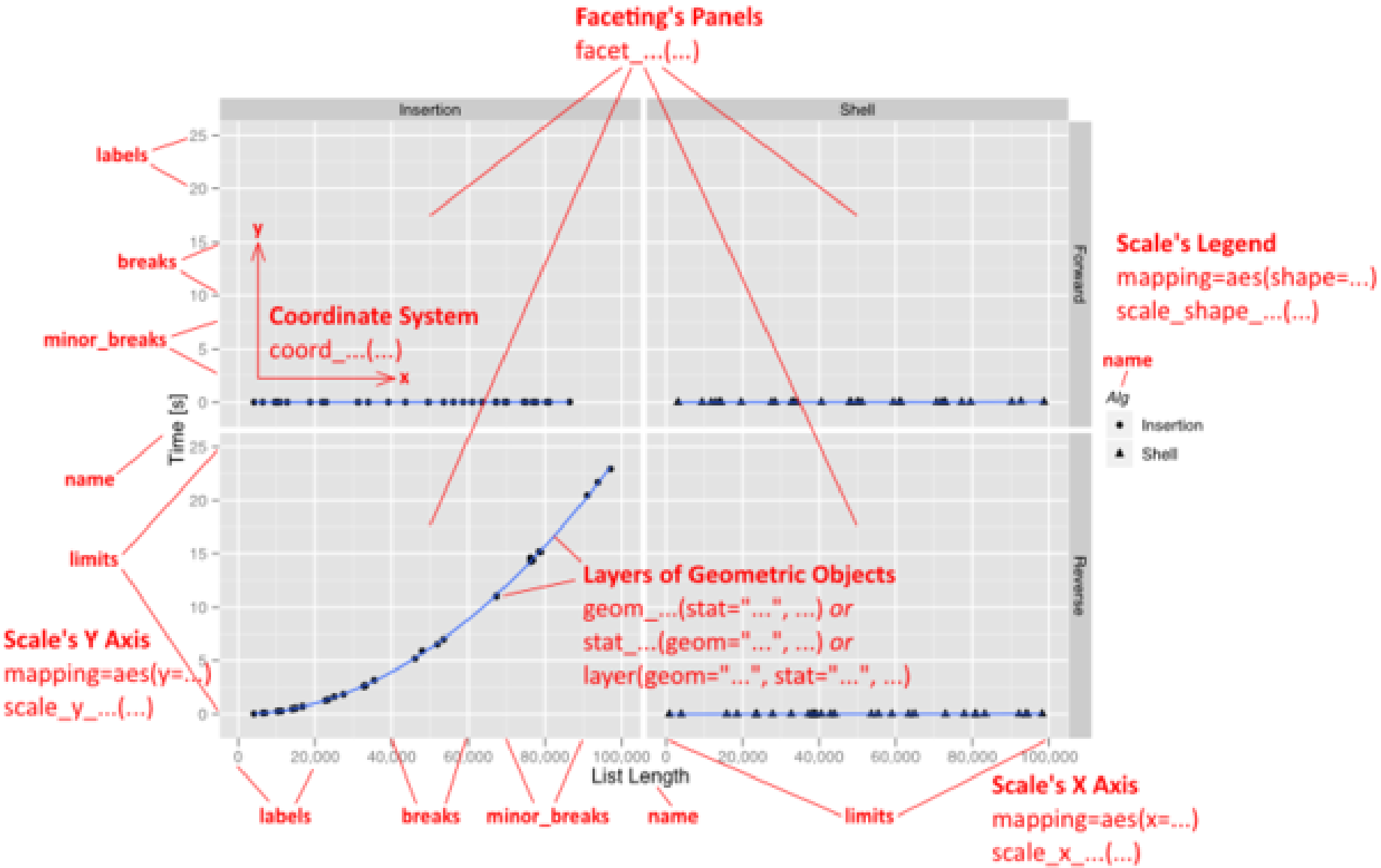


Figure 2. The final graphic, produced by combining the pieces in Figure 1

Ggplot anatomy



Basic plot

```
> head(diamonds)
  carat       cut color clarity depth table price    x    y    z
1  0.23     Ideal     E     SI2  61.5    55   326 3.95 3.98 2.
2  0.21   Premium     E     SI1  59.8    61   326 3.89 3.84 2.
3  0.23      Good     E     VS1  56.9    65   327 4.05 4.07 2.
4  0.29   Premium     I     VS2  62.4    58   334 4.20 4.23 2.
```

```
ggplot(diamonds, aes(x=carat,y=price,color=depth))
+geom_point()
```

- one plot= one dataset (in general)
- aes = aesthetics= what is plotted
- additionnal layers= how to represent the data

GGplot philosophy

1 plot= 1 dataframe

- 1 column= 1 information *dimension*
- Dimensions: `x`, `y`, `shape`, `color`, `fill(ing)`, `facet`, `width`, `linetype`, `alpha`, `(point)size`
- "geom layers"= graphical relations between dimensions
- Continuous (x,y,alpha) $\subset$ Discrete (facet,shape,...)

Layer types

geom_errorbar	Error bars.
geom_histogram	Histogram
geom_line	Connect observations, ordered by x value.
geom_path	Connect observations in original order
geom_point	Points, as for a scatterplot
geom_segment	Single line segments.
geom_text	Textual annotations.
geom_bar	Bars, rectangles with bases on x-axis
geom_polygon	Polygon, a filled path.
geom_raster	High-performance rectangular tiling.
geom_linerange	An interval represented by a vertical line.
geom_bin2d	Add heatmap of 2d bin counts.
geom_boxplot	Box and whiskers plot.
geom_density	Display a smooth density estimate.

Scales

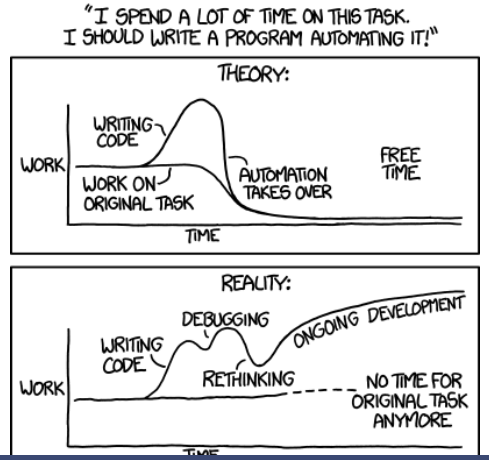
scale_alpha	Alpha scales.
scale_area	Scale area instead of radius (for size).
scale_colour_brewer	Sequential, diverging and qualitative colour scales
scale_colour_gradient	Smooth gradient between two colours
scale_colour_gradient2	Diverging colour gradient
scale_colour_gradientn	Smooth colour gradient between n colours
scale_colour_grey	Sequential grey colour scale.
scale_colour_hue	Qualitative colour scale with evenly spaced hues.
scale_identity	Use values without scaling.
scale_manual	Create your own discrete scale.
scale_linetype	Scale for line patterns.
scale_shape	Scale for shapes, aka glyphs.
scale_size	Size scale.
scale_x_continuous	Continuous position scales .
scale_x_date	Position scale, date
scale_x_datetime	Position scale, date
scale_x_discrete	Discrete position.
labs	Change axis labels and legend titles
update_labels	Update axis/legend labels
xlim(ylim)	Convenience functions to set the limits of the x and y axis.

Efficient R

I'm waiting, why ?

- Premature optimization is the root of all evil (D. Knuth)
- Sometimes you can design your code differently
- Sometimes you should just buy a bigger machine
- But not always...

Figure: Salvador Dali, The Melting Watch, O/C, 1954



Waiting and Waiting

Easiest way not to wait

- is to ask for little..

while coding/exploring your data:

- always work on tractable dataset `sample_n(movies, 100)`
- save intermediary steps that take time
- work in "draft" mode: estimate roughly your distribution before evaluating it more carefully !
- randomized: choose your number of repetitions wisely

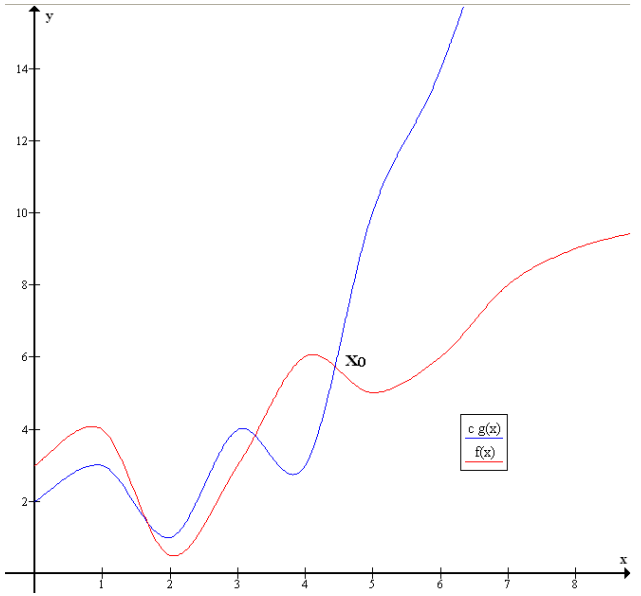
for publication

- Since your code works on small example
- and never depends on sample size
- run it over a week-end !

Complexity

Asymptotic complexity

- Big branch in research
- No need to understand everything. Yet the basics.
- Some problems are inherently harder than others
- How to compare them ? Asymptotic complexity
- $n^{120} = O((1+\epsilon)^n)$
- Asymptotic complexity of your code $\neq$ Asymptotic complexity of the problem (cf sorting algorithms).



R internals

Thanks to Noam Ross, "Vectorization in R, why ?"

To get the most out of R you must understand **how it works:**

$$\begin{bmatrix}1\\2\\3\end{bmatrix} + \begin{bmatrix}1\\2\\3\end{bmatrix} = \begin{bmatrix}2\\4\\6\end{bmatrix}$$

$$1+1=2$$
$$2+2=4$$
$$3+3=6$$

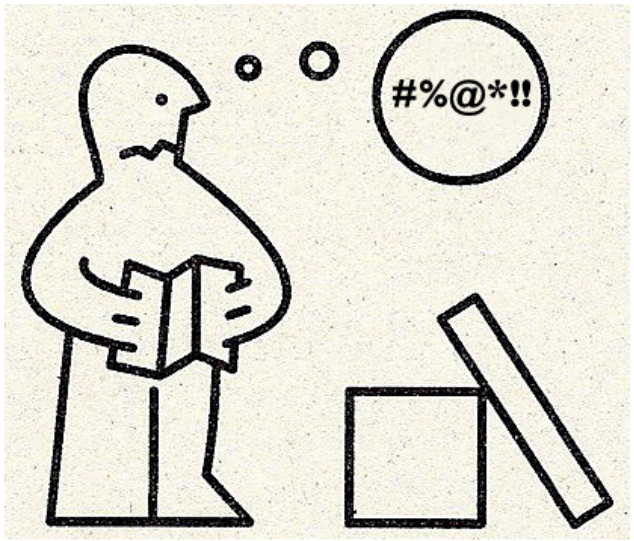
Mathematically the same, yet they don't take the same time:

```
> add=function(i){return(1:i+1:i)}
> add2=function(n){res=c();for(i in 1:n) res[i]=i+i; return(res)}
>  system.time(replicate(100000,add(20)))
   user  system elapsed
  0.664   0.016   0.679
>  system.time(replicate(100000,add2(20)))
   user  system elapsed
  2.928   0.004   2.932
```

R is interpreted

What happens during `i <- 5+5.0` ?

- converts to `"+"(5,5)`
- first argument is an int
- second argument is an int
- What types does my "+" function accepts ?
- Choose the double version, convert first arg
- actually add
- find a place in memory to store the result.
- return a pointer to the result



Whooo ! You don't want to do this more than necessary If R was compiled, much of this would happen *during compilation*

Vectors are homogenous

- One vector= one type
- R figures out the argument types only once !
- $\Rightarrow$(In general) in R fast code is short
- In other languages, you'd better like to split the code in many simple expressions that the compiler can optimise

Memory allocation

- R looks for contiguous place in memory to allocate objects
- If you increase the size of an object, R will have to reallocate
  $\Rightarrow$waste of time !
- $\Rightarrow$preallocate when possible, or use `..ply` functions that preallocate for you.

Preallocation example

```
> noprealloc<-function(n){
   j <- 1;
   for (i in 1:n){
     j[i]=10}
   j}
> system.time(replicate(1e5,
  noprealloc(20)))
   user  system elapsed
  2.816   0.012   2.827
```

```
> prealloc<-function(n){
   j<-rep(NA,n);
   for (i in 1:n){
     j[i]=10}
   j}
> system.time(replicate(1e5
  ,prealloc(20)))
   user  system elapsed
  2.304   0.008   2.309
```

Profile Your Code

Profiling your code= understanding where time is spent

No need to optimise where you don't spend time !

system.time Standard approach, quite inaccurate **microbenchmark**

- More serious approach than replicate
- Tries to minimise system side effects (e.g interleaved execution).
- `library(microbenchmark)`
- `a=microbenchmark(noprealloc(10),prealloc(10), times=1000)`
- `ggplot(a,aes(x=1:nrow(a),y=time,color=expr))+ geom_point()`

profvis

- What costs you time inside a function
- `profvis({ library(ggplot2) g <- ggplot(diamonds, aes(carat, price)) + geom_point(size = 1, alpha = 0.2) print(g) })`

Parallelize !

Many R packages to do parallel operations

Nowadays computer have often many cores

- Parallelization: split your task in multiple subtasks
- Ask each core to do some subtasks

Warning

- Parallelism can be tricky !
- $\Rightarrow$focus on **embarrassingly parallel**(E.P.) chunks: tasks where no communication between the cores is required

Dining philosophers problem: synchronization is pain

doMC package

- **Good news 1:** Many data processing are E.P.
- **Good news 2:** All (clean) lapply functions are E.P.

- `library(doMC)`
- `registerDoMC(cores=4)`
- `[m_dl][m_dl]ply(onwhat,function,.parallel=T)`
- that's it !

Caveats

- Warning: Parallelism works better on computationally expensive tasks.
- Parallelism is not free.
- Parallel $\Rightarrow$harder to debug.
- Some really nasty interactions with GUIs
  (therefore I use emacs)
- You don't need to allocate all your cores to R. (keep some for mail check)

In the **very last case**: you can code in C and interface with R

SQL

MySQL

MySQL is a particular implementation of Ted Codd's RDBs. It is open source, actively and widely supported.
MySQL= a server **and** a set of clients

- `mysqld` is the mysql server
- `mysql` is the standard CLI client
- many "drivers" allow to connect a sql server.
- `RmySQL` is the R driver

Connecting to the server

To connect a server in any case you need

- the server address (host)
- (possibly a port number)
- a set of credentials:
  - user id
  - password
- possiby a database name.

man mysql

- mysql -h host -u user -p [databasename]
- normally you will get a prompt `mysql>`
- (you can setup default credentials in a `.my.cnf` file

MySQL syntax is rigid and painful. Always end your lines with a semicolon

Like in bash and R you can navigate your commands with arrow keys.

Connection Within R

```
library(RMySQL)

con <- dbConnect(dbDriver("MySQL"), user = "ubitrack",
   password = "void", dbname = "ubitrack")

getRequest<- function () {
   req="select * from employees where gender="F")
   a=dbGetQuery(con, req)
   return(a)
}
```

Gathering information in MySQL

Some useful commands:

- `show databases`:to show the list of databases
- `use databasename`: to use a database
- `show tables`: to show the list of tables of the db
- `describe table`: to describe the fields (columns) of a table.
- Ctrl-c once to stop a request evaluation (e.g. too much printing)

Requests

We will only cover information retrieval.

- general format: `select what from table where expression;`
- `expression`= a logical relation involving in general constants and table fields
- `what`= table columns, *, and/or some mathematical operations on table columns (min(col), max(col, sum(col), count(col)), separated by commas.

Some examples

```
mysql> select * from departments;
+---------+--------------------+
| dept_no | dept_name          |
+---------+--------------------+
| d009    | Customer Service   |
| d005    | Development        |
| d002    | Finance            |
| d003    | Human Resources    |
| d001    | Marketing          |
| d004    | Production         |
| d006    | Quality Management |
| d008    | Research           |
| d007    | Sales              |
+---------+--------------------+
9 rows in set (0.00 sec)
```

Some examples

`limit`: limit the number of results

```
mysql> select * from dept_emp limit 4;
+--------+---------+------------+------------+
| emp_no | dept_no | from_date  | to_date    |
+--------+---------+------------+------------+
|  10001 | d005    | 1986-06-26 | 9999-01-01 |
|  10002 | d007    | 1996-08-03 | 9999-01-01 |
|  10003 | d004    | 1995-12-03 | 9999-01-01 |
|  10004 | d004    | 1986-12-01 | 9999-01-01 |
+--------+---------+------------+------------+
4 rows in set (0.00 sec)
```

Some examples

a little summary:

```
mysql> select count(*), min(salary), max(salary),avg(salary) f
+----------+-------------+-------------+-------------+
| count(*) | min(salary) | max(salary) | avg(salary) |
+----------+-------------+-------------+-------------+
|  2844047 |       38623 |      158220 |  63810.7448 |
+----------+-------------+-------------+-------------+
1 row in set (0.83 sec)
```

Some examples

Where clause is a logical predicate on the columns.

```
 select count(*) from employees where gender="M" and day(birth_date)=29 and month(birth_date)=2;
+----------+
| count(*) |
+----------+
|      173 |
+----------+
1 row in set (0.10 sec)
mysql> select count(*) from salaries where salary between 50000 and 110000;
+----------+
| count(*) |
+----------+
|  2134799 |
+----------+
1 row in set (0.57 sec)

mysql> select count(*) from salaries where salary >= 50000 and salary<= 110000;
+----------+
| count(*) |
+----------+
|  2134799 |
+----------+
1 row in set (0.53 sec)
```

Some examples

Count and group make a good team:

```
mysql> select dept_no,count(*) from dept_emp group by dept_no;
+---------+----------+
| dept_no | count(*) |
+---------+----------+
| d001    |    20211 |
| d002    |    17346 |
| d003    |    17786 |
| d004    |    73485 |
| d005    |    85707 |
| d006    |    20117 |
| d007    |    52245 |
| d008    |    21126 |
| d009    |    23580 |
+---------+----------+
9 rows in set (0.11 sec)
```

Some examples

```
mysql> select d.dept_name,e.dept_no,count(*)
  from dept_emp as e inner join departments as d  on
  d.dept_no =e.dept_no group by e.dept_no;
+--------------------+---------+----------+
| dept_name          | dept_no | count(*) |
+--------------------+---------+----------+
| Marketing          | d001    |    20211 |
| Finance            | d002    |    17346 |
| Human Resources    | d003    |    17786 |
| Production         | d004    |    73485 |
| Development        | d005    |    85707 |
| Quality Management | d006    |    20117 |
| Sales              | d007    |    52245 |
| Research           | d008    |    21126 |
| Customer Service   | d009    |    23580 |
+--------------------+---------+----------+
9 rows in set (0.15 sec)
```

Version Management

Sorry ! No time for this.
But here is a very nice tutorial:

`http://rogerdudler.github.io/git-guide/index.fr.html`

Publication (considerations)

Pdf $\neq$ Png

To save a picture in ggplot:

- last picture: `ggsave("file.pdf")`
- any picture stored: `ggsave(p,file="file.pdf")`
- Warning: you can set `width=` and `height=`
  by default: format of your plot window.
- Png (and jpg)=a matrix of pixels, each entry is a color = raster image
- Pdf (and eps, svg)=a generic container that can describe *both* vector graphics and raster images

Impact

- Vector graphics= often small, **no resolution loss**
- In general, plots=pdfs
- Do not generate png and then convert in pdf !

Choose the plots to polish

- You will only present some of your plots in a paper
- Polishing a plot requires time: use it wisely
- I've never submitted a paper without re-plotting something <24hrs to the deadline
- $\Rightarrow$make sure this plots are easily repeatable (writeRDS,comments...)
- Towards repeatable science: be ready to diffuse your code !

If the idea of re-running your experiments makes you inconfortable, you've done something wrong.

The plots you'll present

Plots should "tell a story" !

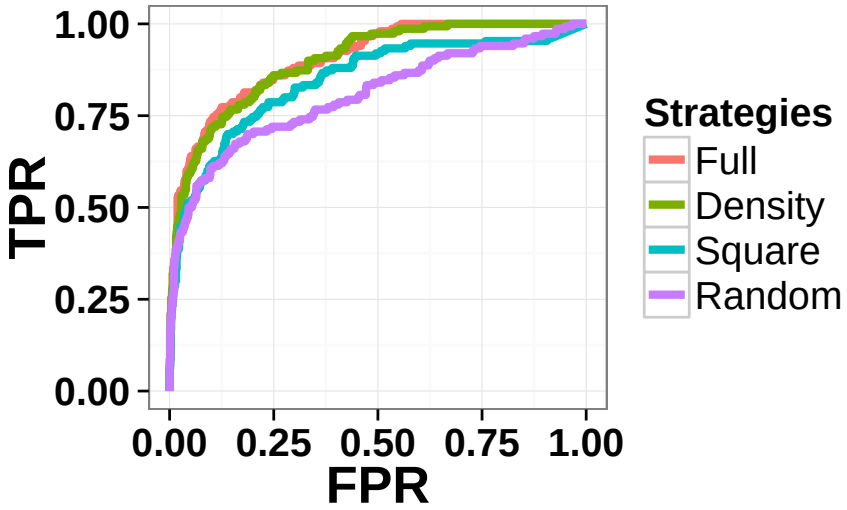
- They should be coherent together:
  - they should not contradict !
  - same naming scheme, same color scheme for same objects
  - same aspect ratio (and resolution)
- Constants homogeneity: allow the reader to "correlate" your plots!
- People first look at plots, then text !
- Metrics: be sure people get them easily (*e.g.* bigger is better)

Readability

- 7-10% colorblind mens
- $\approx$ 50% of researchers wear glasses.
- Still some Black and White printers around.

Describing your plots

- (obvious) Always add a legend to your figures
- Always refer to a figure in the text
- Don't rephrase your legend in the text.
- Legend = what is it, text= why it is important.
- Protip: in the text, take a (x,y) point and explain what it means.



Overplotting

Once you get used, plotting is dangerously easy

- Plotting is not researching
- Like powerpoints, they are "rewarding"... and time consuming
- Before plotting, always think about what you expect
- Question the plot you then see:
  - Did you expect that ?
  - Why so ? Why is it different ?

Concluding Remarks

Key Takeaways

- EDA $\neq$ Confirmatory: Detective work vs. Jury trial
- Think before you plot
- R is a wonderful tool/ecosystem
- Target repeatability and structure
  - Keep a high level approach
  - While understanding the details
- Performance:
  - Know what to optimise (and whether to optimise or not)
  - Nicely designed code is often naturally optimal in R

This is the end

Philosophy of the course

- Dense content, take time to chew
- Being fully efficient requires a bit of practice
- Let all this cool a bit and try to adapt to your research
- Explore around ! Lots and lots haven't been mentionned

- Ploting is not all
- Clicking is cheating!

Thank you for attending!

Playing with Data

Same problem, different viewpoints

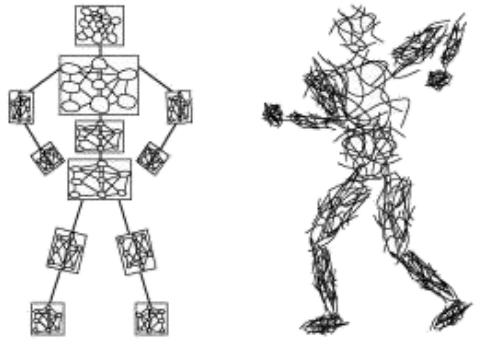
- Information Retrieval= classification and indexing
- Artificial Intelligence= classical term (obj. is to *replicate intelligence*)
- Machine Learning= build programs that improve automatically through experience.
  - unsupervised learning = patterns in input stream
  - supervised learning = logical regression + classification
  - (+reinforcement learning = continuous but relax training)

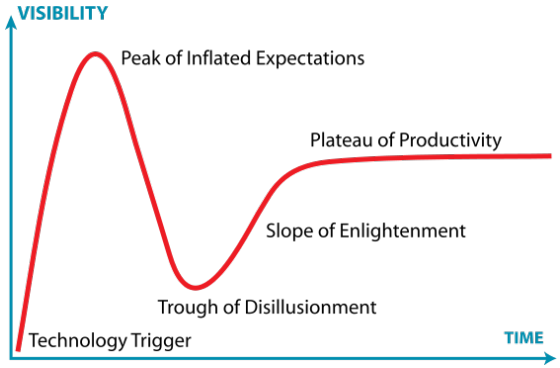
Alan Turing
1912-1954

Edsger Dijkstra
1930-2002

A brief history of AI times

- 1952-1956 The birth of artificial intelligence
- 1956 Dartmouth Conference, birth of AI
- 1956-1974 The golden years
- 1974-1980 1^{st} AI winter (Moravec's paradox, *Perceptrons*)
- 1980-1987 Boom
- 1981 The money returns: the 5^{th} generation project
- 1987-1993 Bust: the second AI winter
- 1993-2001 AI
- 2000-present learning, big data and artificial general intelligence:





Tools

- Linear regression
- Nearest neighbor
- SVM = support vector = linear classifiers
- logical regression
- Neural Networks
- Bayesian networks= conditional probabilities on a graph where each vertex is a random variable
- Clustering
- Decision tree

Training

The Elements of Statistical Learning (2nd edition) Hastie, Tibshirani and Friedman (2009). Springer-Verlag. (763 pages)

General Learning framework =

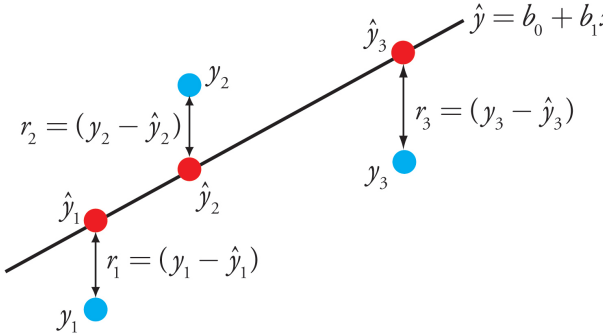
- One observation = p dimensions (*e.g.* Age, income, size, IsMarried, carLength)
- We need many (N) observations $x_1, \ldots, x_N$ to train our model.
- Our goal is to *predict* an output value Y (*e.g.* shoeSize, retirementAge) or a group G (isHappy, SES).
- We are given (x_i, y_i) or (x_i, g_i) couples from which we train a model= a function f s.t. $\hat{Y} = f(X)$

Linear Regression

- We seek the best **linear** combination of variables to predict output
- $\hat{Y} = \beta_0 + \sum_{i=1}^{p} x_i . \beta_i$
- β_0 = constant term= intercept
- $Y = X^T \beta$ is we "force" the constant 1 in X.

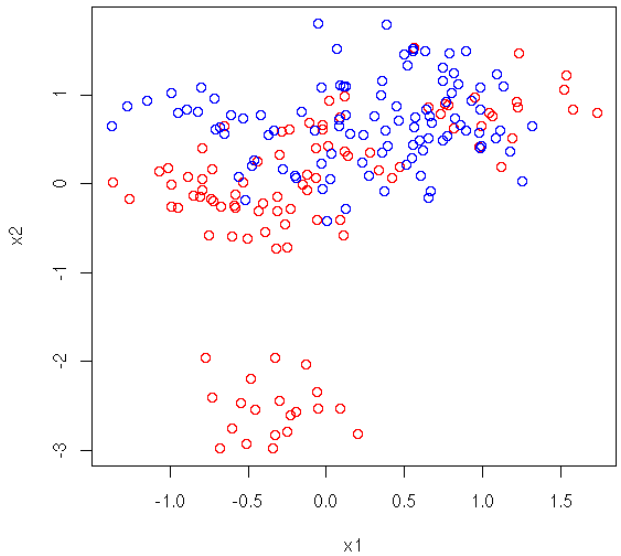
Fitting β ?

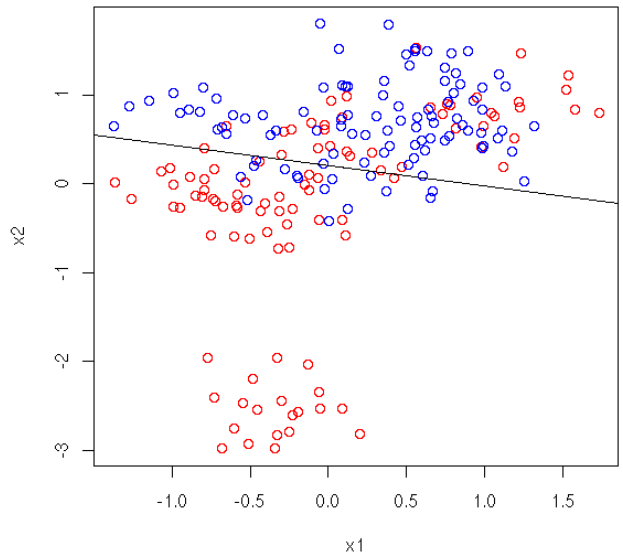
- A good β = A β that makes few errors
- $\Rightarrow$Define error function
- $\Rightarrow$fitting β = minimisation problem.
- Popular error function= Root Mean Square Error
- $RSS(\beta) = \sqrt{\sum_{i=1}^{N} (y_i - x_i^T \beta)^2}$
- (if we're lucky) $\hat{\beta} = (X^T X)^{-1} X^T y$

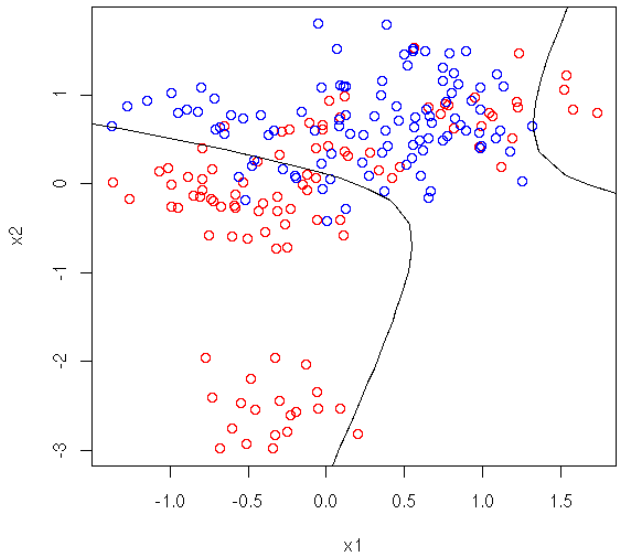


Linear Regression for Classification

- Regression on a binary variable $y \in \{0, 1\}$
- Very simple classifier
- Membership $= \hat{y} < 0.5$



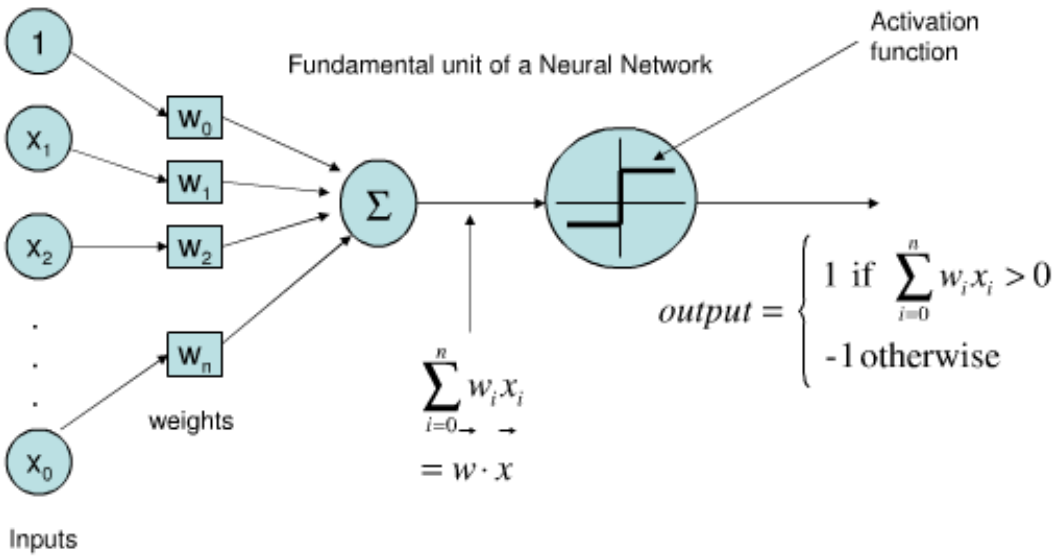




Perceptron

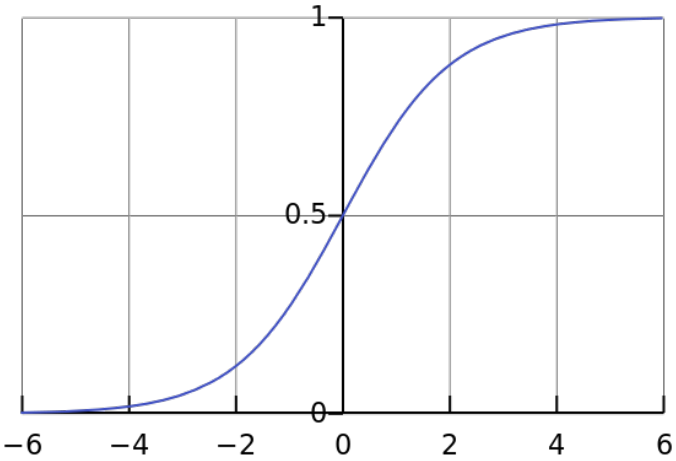
- Fundamental building block of an ANN
- Rosenblatt, 1957
- Trained by a Hebb-rule like approach:

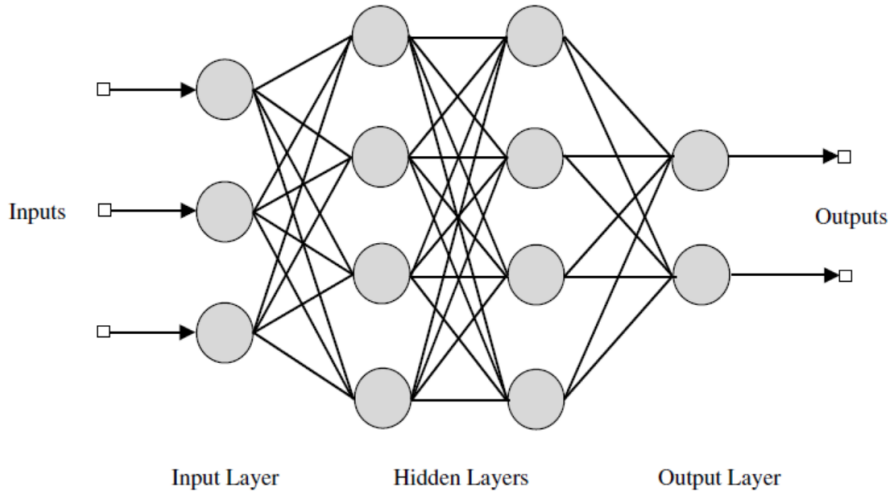
$$W_i' = W_i + \alpha(Y_t - Y)X_i$$



To Multilayer Neural Networks

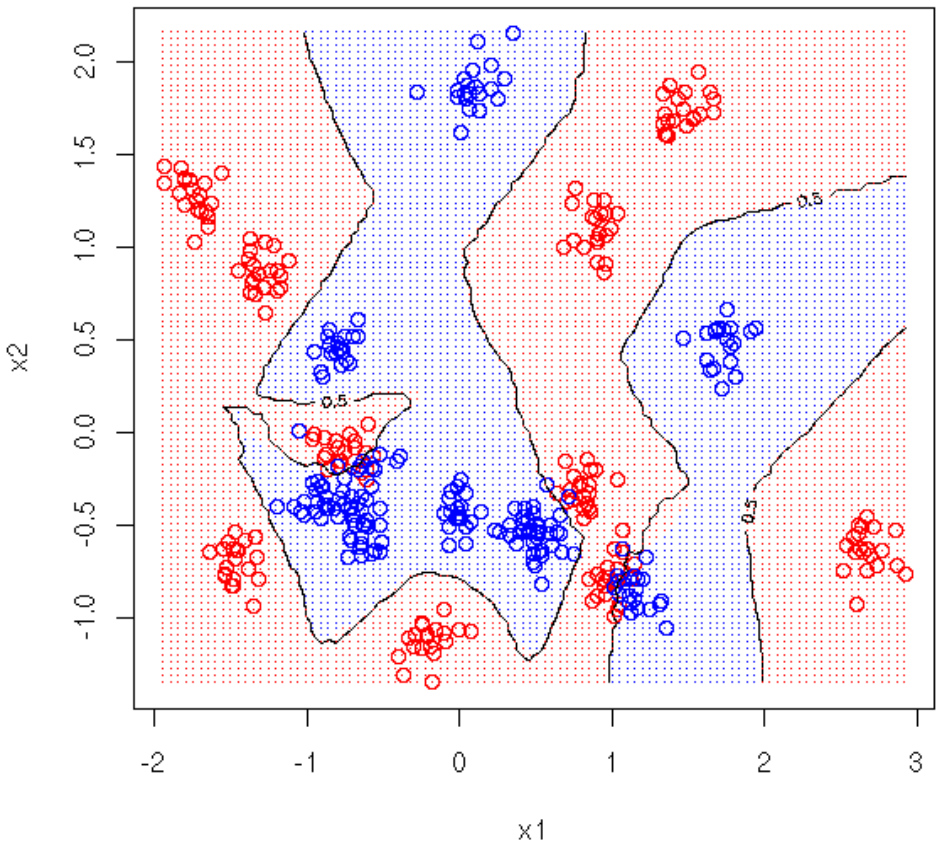
- Put many perceptrons together
- How to train them ? Backpropagation (="gradient descent")
- But need a differentiable error function...
- **Universality - Cybenko 89** a feed-forward network with a single hidden layer containing a finite number of neurons, can approximate continuous functions on compact subsets of $\mathbb{R}^n$





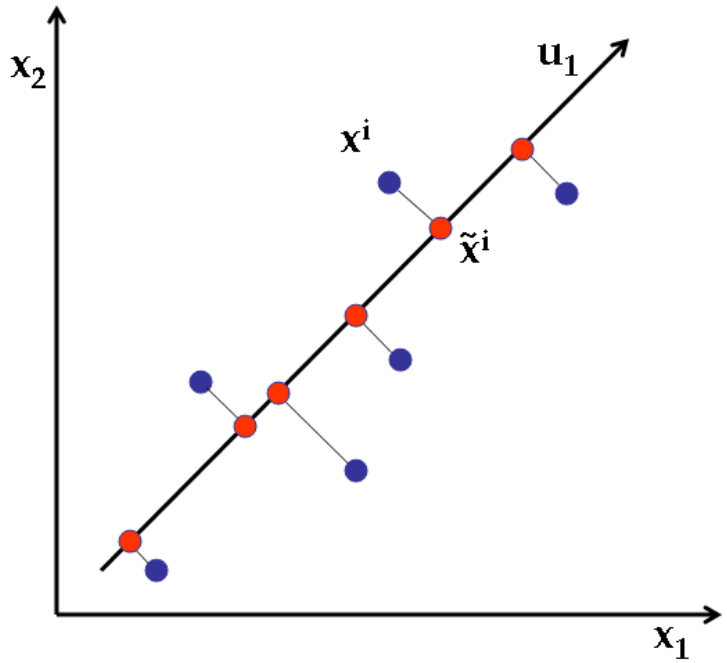
Nearest Neighbors

- We seek to model Y as a function of its neighbors
- Intuition: let's take the "most similar" already encountered cases, and try to extract a consensus.
- =assumes a locally constant function
- $\hat{Y}(x) = 1/k \sum_{x_i \in N_k(x)} y_i$
- heavily used in recommendation
- Impact of k



PCA

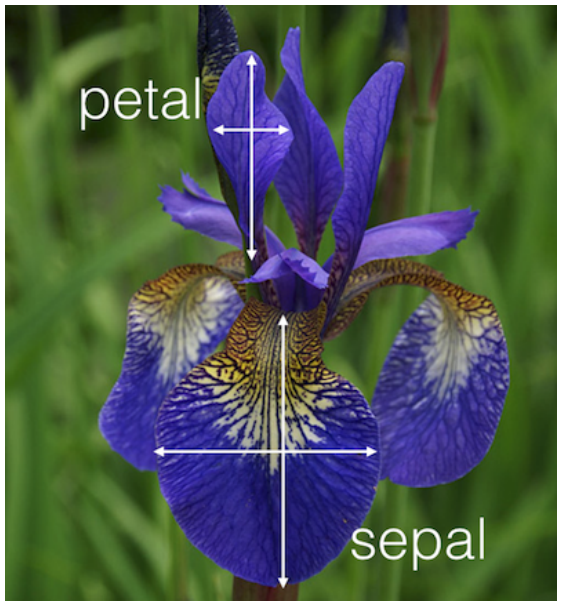
- Factorization method.
- Objective = "compress data"
- =Reduce number of dimensions while maximising information kept
- = maximise variance
  $V = 1/n \sum (x_i - \mu)(x_i - \mu)^T$.
- $\Rightarrow$eigenvalues $\sigma_1, \ldots \sigma_p$
- keep associated eigenvectors



Linear Discrimant Analysis

- Data-mining perspective: find a classifier
- Geometric perspective: transform the data = project on a new basis.
- = Find a space that allows to distinguish between groups

- Spread = Variance-Covariance matrix
- ..Within groups = $W = \frac{1}{n} \sum_k n_k \times W_k$
- ..Between groups = $B = \frac{1}{n} \sum_k n_k ({}^t(\mu_k - \mu))(\mu_k - \mu)$
- Huyghens says : Total variance $V = B + W$
- Objective = find a projection axix u s.t.: $u = argmax \frac{u_1' B u_1}{u_1' V u_1}$

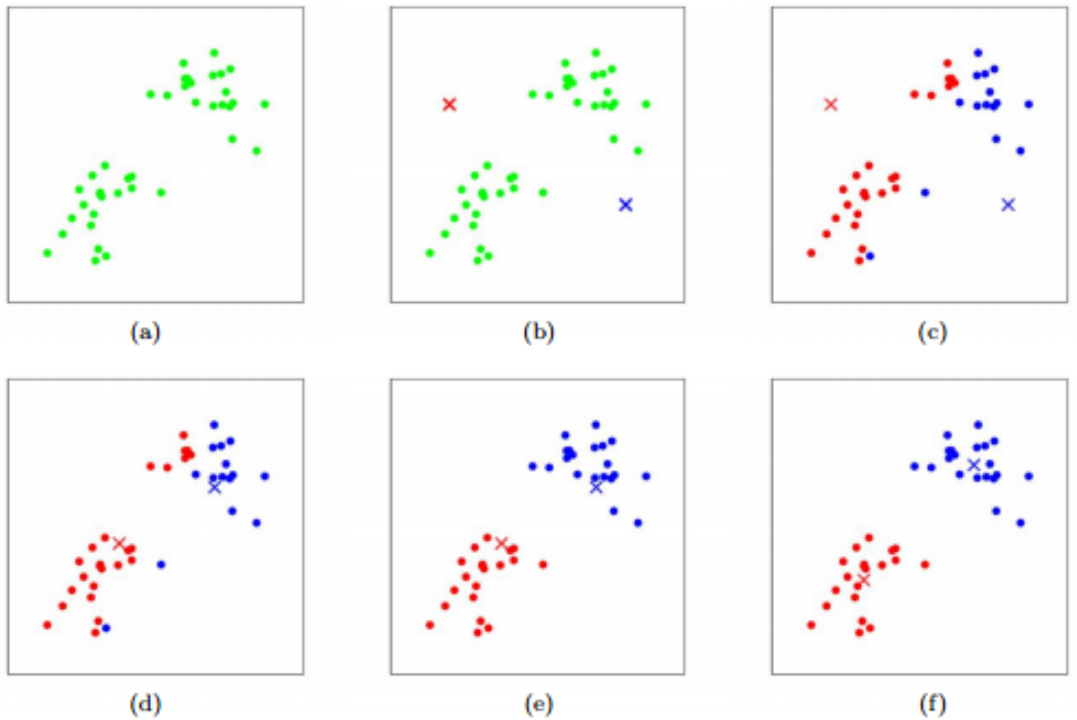


Clustering/k-means

- partition the k observations into k sets $S_1 \dots S_k$
- minimize the within-cluster sum of squares
- Optimal solution= NP
- Practical iterative approach

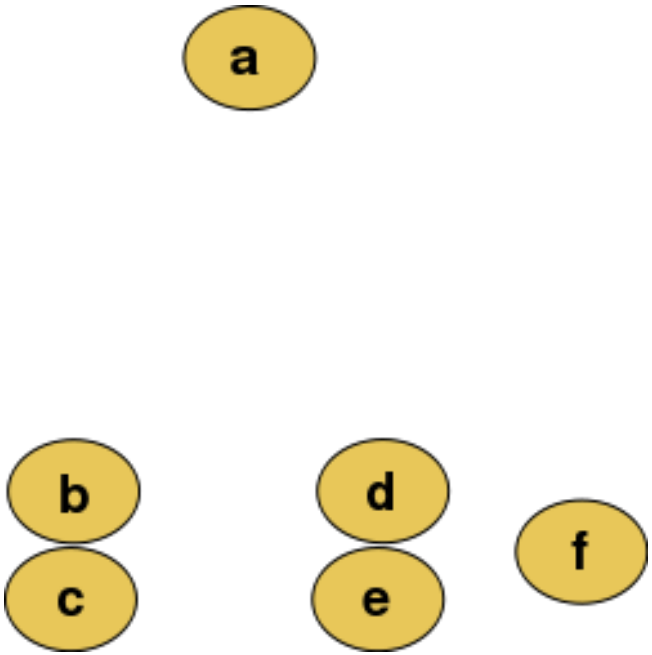
$$\arg\min_{\mathbf{S}} \sum_{i=1}^{k} \sum_{\mathbf{x} \in S_i} \|\mathbf{x} - \boldsymbol{\mu}_i\|^2$$

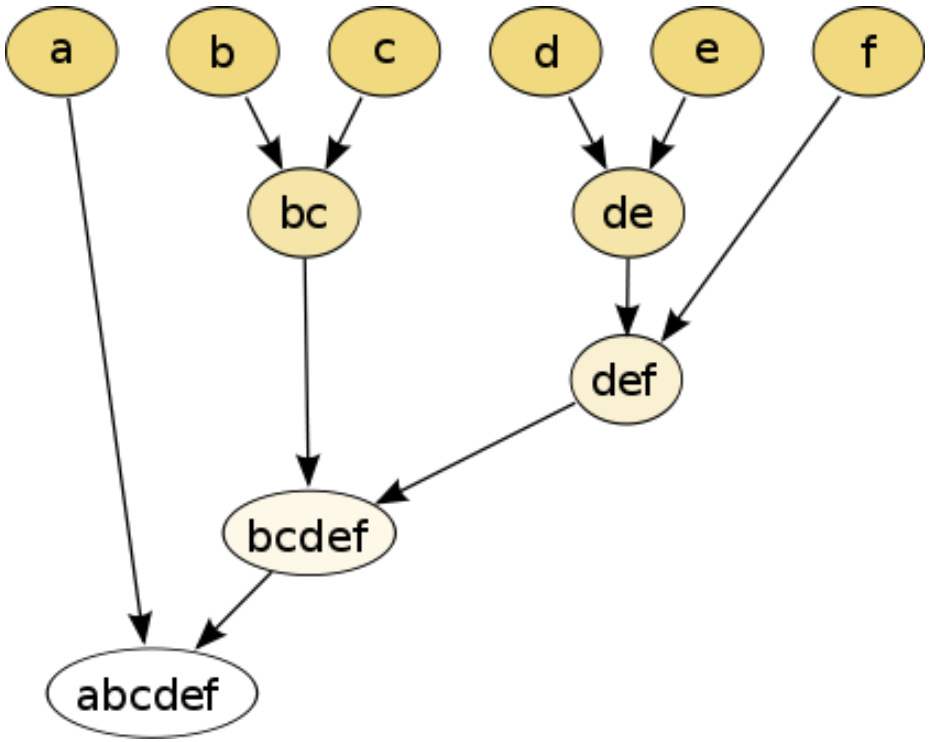
/!\local minima



Clustering/ward

- Hierachical/Agglomerative clustering
- Principle:
  - Choose a distance **d** function between two clusters
  - Start with each element in its own cluster
  - Pick the 2 closest and merge them
  - Only one shall remain.





Testing

- Is my model good ?
- What is the best model ?
- Several approaches:
  - Statistical hypothesis testing
  - Prediction Capability

Central question = how will my model perform for real ?

Crossfold Validation

- Split dataset in 2: Train and Test
- How to minimise the effect of this randomization ?
- Split dataset in 10, select each one as test and train on the 9 others, average error.

Overlearning

- Danger of all learning = failure to generalize
- Never train and test on the same set !
- ... yet compare prediction accuracy on test and training.

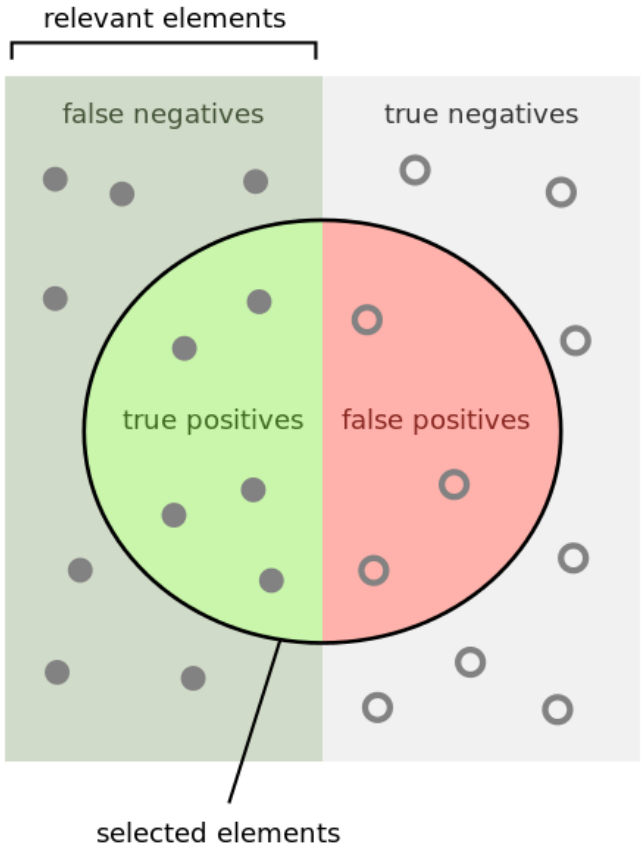
Testing Classification

- How to count classification errors
- *E.g.* detect spam, find revelevant webpages, detect terrorists
- Precision=

$$(CorrectAnswers)/(TotalAnswers)$$

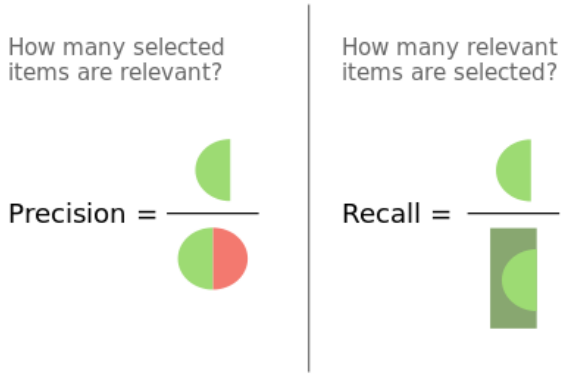
- Recall =

$$(CorrectAnswers)/(TotalCorrect)$$



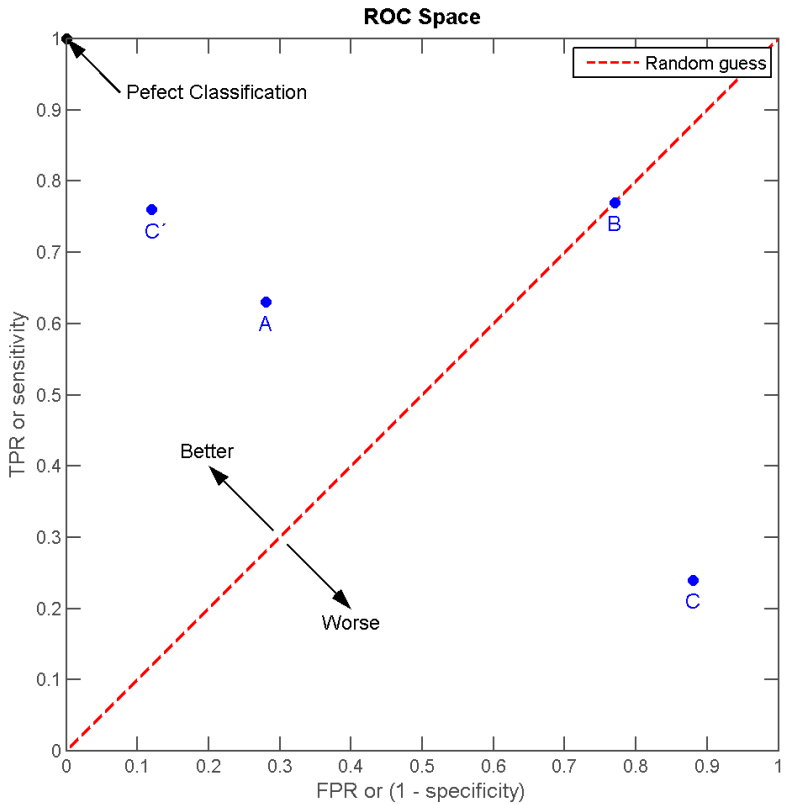
Several variations

- Sensitivity (=TPR=Recall)
- Specificity (=TNR=True Negative Rate)
- Type I/II errors...



Roc Curves

- Usually, detection associated with a score/confidence E.g. $P(Y = 1|X)$
- A threshold T parametrizes decision: $\hat{Y} = 1 \Leftrightarrow P(Y = 1|X) > T$
- $T > T' \Rightarrow$ $FPR(T) < FPR(T'), TPR(T) < TPR(T')$
- Roc curves summarize classifier behaviour wrt T.
- AUC= Area under curve $= \int_0^1 f(T)dT$



Logistic Regression

Bonus