



# Simulation and Performance Evaluation

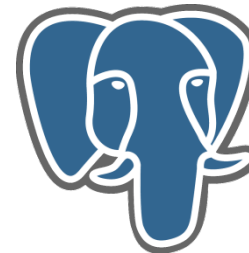
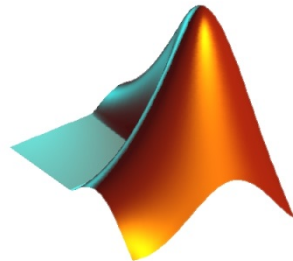
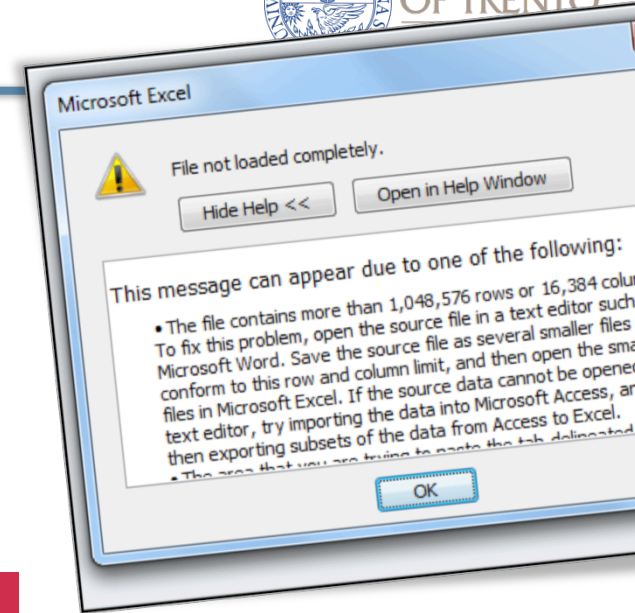
## Basic R Usage

Michele Segata, Renato Lo Cigno

Original version taken from the Network Simulation course held at the University of Paderborn, Germany

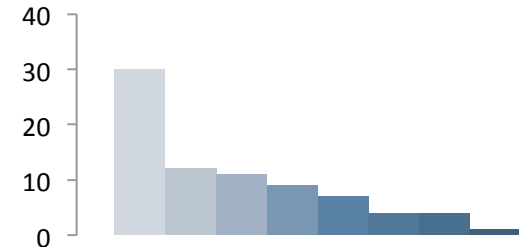
Special thanks to Falko Dressler and Christoph Sommer

- How to analyze large amounts of data?
  - SAS
  - SPSS
  - Stata
  - Matlab
  - SciPy (or Pandas, or ...)
  - SQL
  - Excel 🤔
  - ...
  - R





- Why R?
  - Oldest
    - Implementation of S, designed at Bell Labs starting 1975
  - Newest
    - Actively maintained by international team
    - Version 3.2.5 released in April 2016)
    - Version 3.3.1 available since June 2016
  - Open Source
    - Available for every major OS
  - Popular
    - Consistently ranks among top 3 software tools for data analysis





- What is R?
  - the computer language R
  - an interpreter of code written in R
  - a execution environment that contains the R interpreter
- From Wikipedia:
  - “R is an implementation of the S programming language combined with lexical scoping semantics inspired by Scheme. S was created by John Chambers while at Bell Labs. R was created by Ross Ihaka and Robert Gentleman at the University of Auckland, New Zealand, and is currently developed by the R Development Core Team, of which Chambers is a member. R is named partly after the first names of the first two R authors and partly as a play on the name of S.”
- R is a (kind of) scripting language
  - Automatic variables, garbage collection
  - Lots of “smart” shortcuts (for better or worse)

```
> 1 + 2
[1] 3
> exp(2i*pi)
[1] 1-0i
> mean(c(0, 2, 3))
[1] 1.666667
> round(2.5)
[1] 2
> round(3.5)
[1] 4
```



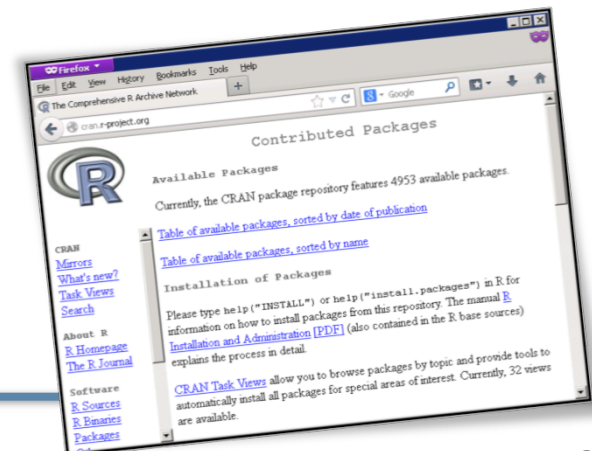
- To install R, you can do the following depending on your OS:
  - **Linux:** Just open your terminal and type
    - `sudo apt-get install r-base`
  - **Mac OS X:** If you have MacPorts installed, open your terminal and type
    - `sudo port install R`
  - **Mac OS X:** If you don't have MacPorts, you can download R binaries from the official R website (<http://www.r-project.org/>)
  - **Windows:** Download the binaries from the official R website

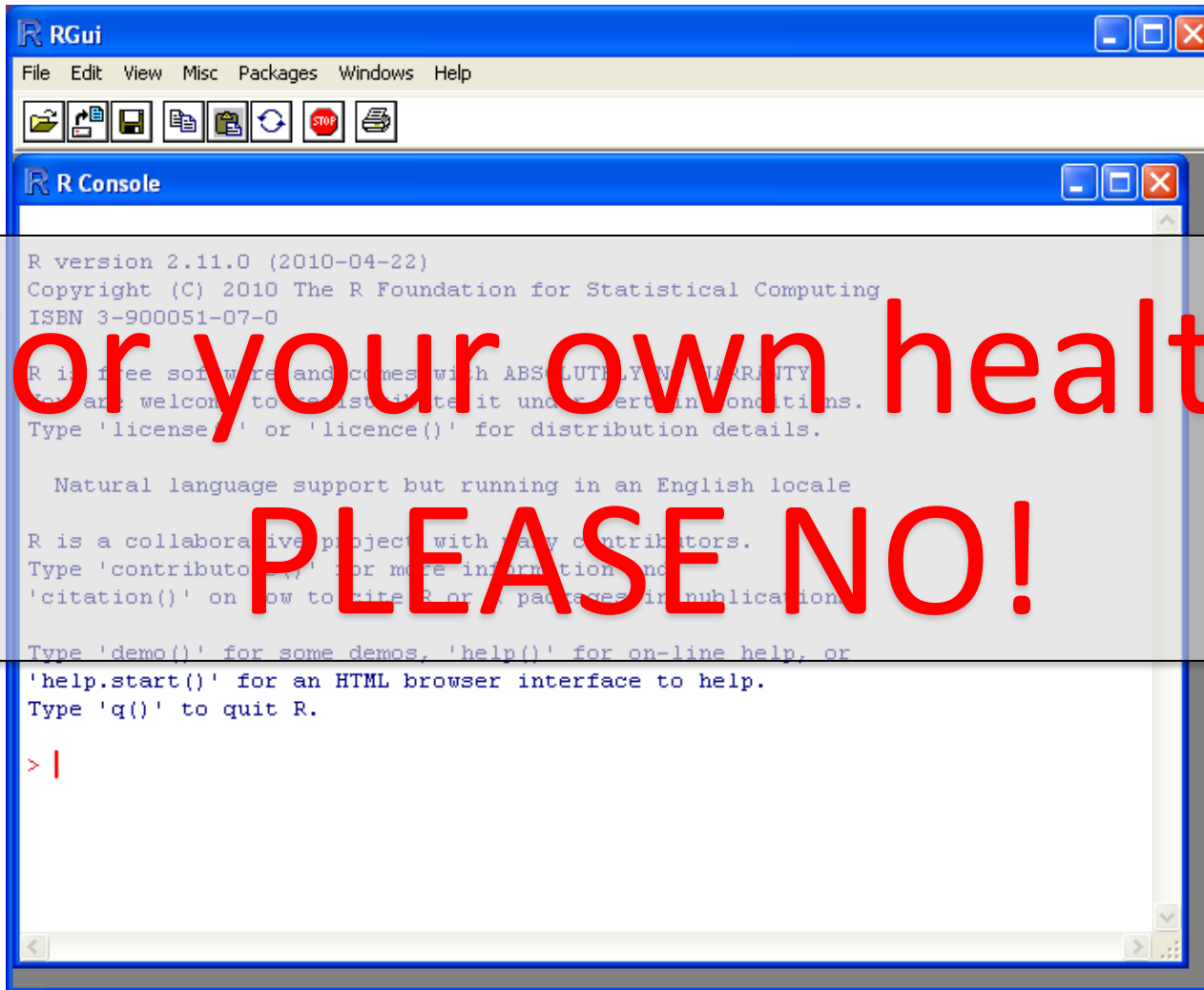
- Frontend

- Official frontend: [www.r-project.org](http://www.r-project.org)
  - R
  - Rscript (to run R scripts from command line)
- Or any of
  - RStudio, Revolutions, Tinn-R, Deducer, RKward, R Commander, Vim R, ...

- Packages

- Extend R functionality
- Written in C, Java, Fortran, or R
- Official repository: [cran.r-project.org](http://cran.r-project.org)
  - Comprehensive R Archive Network (CRAN)
- Or any of
  - Bioconductor, Crantastic, R-Forge, ...





The screenshot shows the RStudio interface with the following components:

- Source Editor:** Contains R code for loading data, summarizing it, and creating a scatter plot.
 

```

1 library(ggplot2)
2 source("plots/formatPlot.R")
3
4 view(diamonds)
5 summary(diamonds)
6
7 summary(diamonds$price)
8 ave_size <- round(mean(diamonds$carat), 4)
9 clarity <- levels(diamonds$clarity)
10
11 p <- qplot(carat, price,
12            data=diamonds, color=clarity,
13            xlab="Carat", ylab="Price",
14            main="Diamond Pricing")
15
16 format.plot(p, size=24)
      
```
- Console:** Shows the output of the executed code, including summary statistics for the 'diamonds' dataset.
 

```

x           y           z
Min.   : 0.000   Min.   : 0.000   Min.   : 0.000
1st Qu.: 4.710   1st Qu.: 4.720   1st Qu.: 2.910
Median : 5.700   Median : 5.710   Median : 3.530
Mean   : 5.731   Mean   : 5.735   Mean   : 3.539
3rd Qu.: 6.540   3rd Qu.: 6.540   3rd Qu.: 4.040
Max.   :10.740   Max.   :58.900   Max.   :31.800
> summary(diamonds$price)
  Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
  326   950   2401   3933   5324  18820
> ave_size <- round(mean(diamonds$carat), 4)
> clarity <- levels(diamonds$clarity)
> p <- qplot(carat, price,
+           data=diamonds, color=clarity,
+           xlab="Carat", ylab="Price",
+           main="Diamond Pricing")
>
> format.plot(p, size=24)
> |
      
```
- Workspace:** Shows the 'diamonds' dataset with 53940 observations and 10 variables.
- Plots Panel:** Displays a scatter plot titled 'Diamond Pricing' showing 'Price' on the y-axis (0 to 15000) and 'Carat' on the x-axis (0.0 to 3.5). The points are colored by 'Clarity', with a legend on the right showing categories: I1, SI2, SI1, VS2, VS1, VVS2, VVS1, and IF.

**MORE FREE TIME!**





# R Frontends – Vim R Plugin



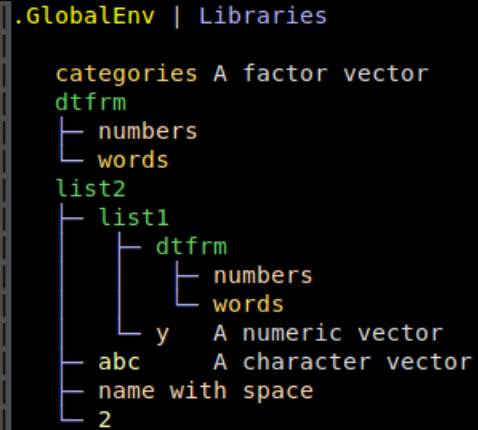
```
example.R (/tmp) - VIM
numbers <- 1:3
words <- c("word1", "word2", "word3")
categories <- as.factor(words)
dtfrm <- data.frame(numbers, words)

attr(numbers, "label") <- "A numeric vector"
attr(words, "label") <- "A character vector"
attr(categories, "label") <- "A factor vector"

list1 <- list(dtfrm = dtfrm, y = numbers)
list2 <- list(list1 = list1, abc = words)
list2$name with space <- 1:10
list2$`2` <- c("one", "two")
list3 <- list(abc = categories, list1 = list1)
rm(list1)

example.R [+] 1,1 All Object_Browser 8,1 Top

> list1 <- list(dtfrm = dtfrm, y = numbers)
> list2 <- list(list1 = list1, abc = words)
> list2$name with space <- 1:10
> list2$`2` <- c("one", "two")
> list3 <- list(abc = categories, list1 = list1)
> rm(list1)
> source('/home/jakson/src/Vim-R-plugin/r-plugin/vimbrowser.R') ; .vim.browser()
>
```



- Assigning data
  - `x <- 1`
  - `x = 1` also works
  - `x <- (1+2)*NA*3` ⇨ `x <- NA`
- Creating data of specific type and class
  - `v <- c(10, 20, 30, 40, 50, 60)`
  - `l <- list(a="apple", b="balloon", c="crayons")`
  - `m <- matrix(v, nrow=3)`
  - `d <- data.frame(student=c("alice", "bob", "carol"), grade=c(1,2,3))`

	x
1	10
2	20
3	30
4	40
5	50
6	60

a	b	c
apple	balloon	crayons

	v1	v2
1	10	40
2	20	50
3	30	60

	student	grade
1	alice	1
2	bob	2
3	carol	3



- Printing

- `print(v)` ⇒ [1] 10 20 30 40 50 60
- `summary(v)` ⇒ Minimum: 10.0 1<sup>st</sup> Quartile: 32.5 Median: 45.0 ...
- ...

	x
1	10
2	20
3	30
4	40
5	50
6	60

a	b	c
apple	balloon	crayons

	v1	v2
1	10	40
2	20	50
3	30	60

	student	grade
1	alice	1
2	bob	2
3	carol	3



- At its core, an R object is defined by three properties
  - Type
    - homogeneous vector types (logical, integer, double, ...)
    - heterogeneous list type (list)
    - miscellaneous types (function, expression, ...)
  - Length
    - Only really sensible for vectors and lists
  - Attributes
    - class (matrix, factor, data.frame, ...)
      - determines how generic functions interoperate
    - dim (dimensions of matrix)
    - levels (possible values of factor)
    - names, dimnames, rownames, colnames, ...
    - comment
    - ...



- Subset operator (attn: R is one-based, not zero-based)
  - `print(v[ c(2, 3, 4) ])` ⇒ `[1] 20 30 40`
  - `print(v[2:4])` ⇒ `[1] 20 30 40`
  - `print(v[c(T,F,F,F,F,T)])` ⇒ `[1] 10 60`
  - `print(v[-2])` ⇒ `[1] 10 30 40 50 60`
  - `v[2] <- 99; print(v)` ⇒ `[1] 10 99 30 40 50 60`
  - `print(m[1,2])` ⇒ `[1] 40`
  - `print(m[1,])` ⇒ `[1] 10 40`
  - `print(m[,2])` ⇒ `[1] 40 50 60`

	x
1	10
2	20
3	30
4	40
5	50
6	60

a	b	c
apple	balloon	crayons

	v1	v2
1	10	40
2	20	50
3	30	60

	student	grade
1	alice	1
2	bob	2
3	carol	3



- Element operator
  - `print(l[["a"]])` ⇨ `[1] "apple"`
  - `print(d[["grade"]])` ⇨ `[1] 1 2 3`
- CAREFUL: `d[["grade"]]` IS DIFFERENT FROM `d["grade"]`
- Name operator
  - `print(l$a)` ⇨ `[1] "apple"`
  - `print(d$grade)` ⇨ `[1] 1 2 3`

	x
1	10
2	20
3	30
4	40
5	50
6	60

a	b	c
apple	balloon	crayons

	v1	v2
1	10	40
2	20	50
3	30	60

	student	grade
1	alice	1
2	bob	2
3	carol	3



- Defining functions
  - `f <- function(x, y=1, offset=0) { offset + x * y }`
  - `print(f) ⇨ function (x, y = 1, offset = 0) { offset + x * y }`
- Calling functions
  - `f(3, 7, 1000) ⇨ [1] 1021`
  - `f(3, 7) ⇨ [1] 21`
  - `f(offset=1000, x=3, y=7) ⇨ [1] 1021`
  - `f(3, offset=1000) ⇨ [1] 1003`

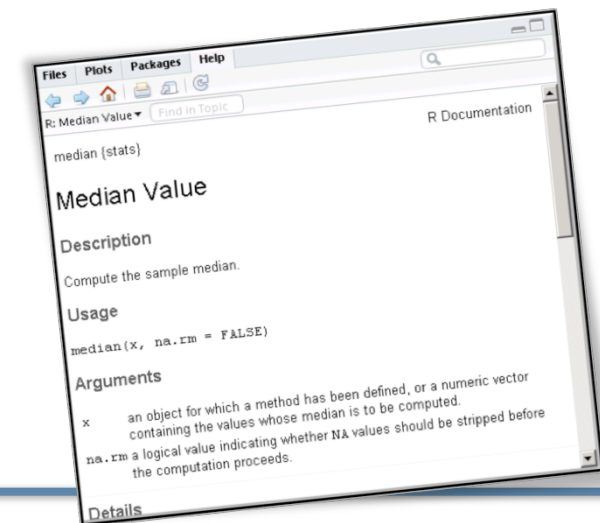
- Basic statistics

- `X <- c(5, 20, 40, 50, 55, 70, 80, 80, 900)`
- `Y <- c(10, 20, 30, 40, 50, 60, 70, 80, 90)`
- `mean(X)` ⇒ 144.4444
- `median(X)` ⇒ 55
- `quantile(X, 0.25)` ⇒ 40
- `glm.fit(1:9, Y, intercept=F)$coefficients` ⇒ 10
- `wilcox.test(X, Y)$p.value` ⇒ 0.72

	X	Y
1	5	10
2	20	20
3	40	30
4	50	40
5	55	50
6	70	60
7	80	70
8	80	80
9	900	90

- Help and more information

- `help('mean')`
- `?mean`
- `citation('vioplot')`







- Creating
  - `d <- data.frame(student=rep(c("alice", "bob", "carol"),4), points=rep(c(90,30,70,50,40,10), 2))`
- Grouping averages
  - `a <- aggregate(d$points, by=list(who=d$student), FUN=mean)`
  - or use `ddply` by `plyr` (see example)
- Adding a new column
  - `a$grade <- ifelse(a$x >= 50, "pass", "fail")`

	student	points
1	alice	90
2	bob	30
3	carol	70
4	alice	50
5	bob	40
6	carol	10
7	alice	90
8	bob	30
9	carol	70
10	alice	50
11	bob	40
12	carol	10

	who	x
1	alice	70
2	bob	35
3	carol	40

	who	x	grade
1	alice	70	pass
2	bob	35	fail
3	carol	40	fail



- Syntax:
  - `install.packages("package name")` or
  - `install.packages(vector of packages names)`
  - e.g.: `install.packages(c("plyr", "moments"))`

There is no need of manually downloading, compiling, and installing packages!

- R will ask you which mirror to use, download, and install the packages
  - with dependencies
- To load your library:
  - `library("plyr")` or `require("plyr")`

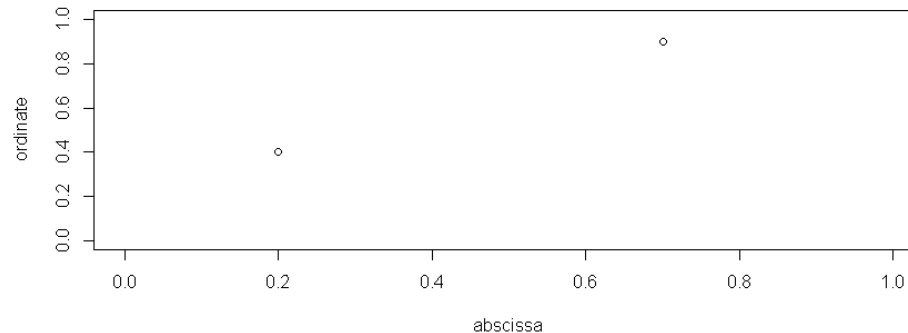


- Read data frame from text file
  - `d <- read.csv('data.csv')`
  - `d` will be a `data.frame` where column names matches csv column names

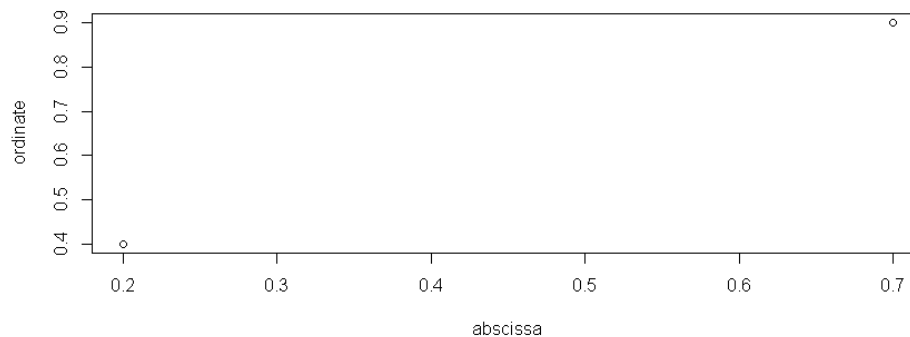
	student	points
1	alice	90
2	bob	30
3	carol	70
4	alice	50
5	bob	40
6	carol	10
7	alice	90
8	bob	30
9	carol	70
10	alice	50
11	bob	40
12	carol	10



- R can do plots, too
- Low level plotting commands
  - `plot.new()`
  - `plot.window(xlim=c(0,1), ylim=c(0,1))`
  - `points(c(0.2, 0.7), c(0.4, 0.9))`
  - `box()`
  - `axis(1)`
  - `axis(2)`
  - `title(xlab="abscissa", ylab="ordinate")`

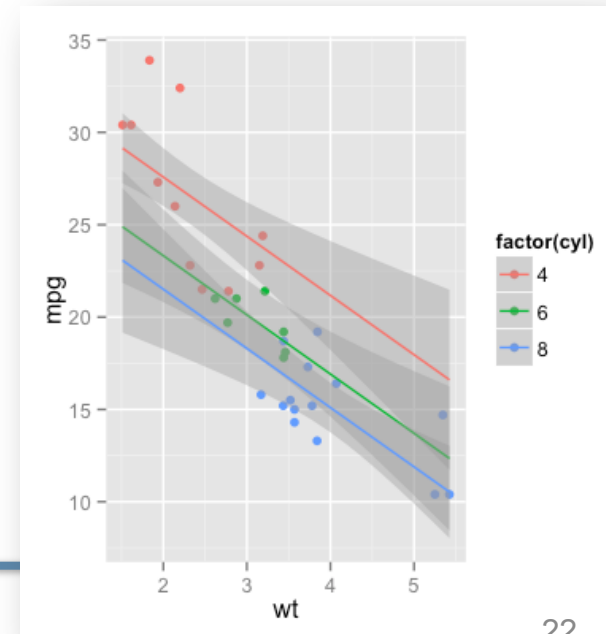


- High level plotting commands
  - generic “plot” command guesses good plot type
  - `plot(c(0.2, 0.7), c(0.4, 0.9), xlab="abscissa", ylab="ordinate")`
- Can mix high level and low level plotting commands
  - e.g., annotations, lines, points, `abline`, ...

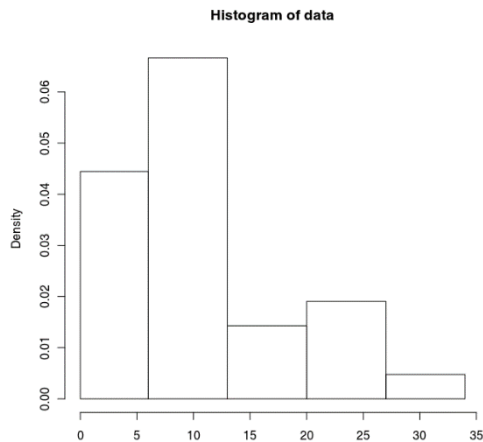


- Tons of packages for basically EVERYTHING
- **Lattice**
  - Good for *conditioning* types of plots (a vs. b depending on c...)
  - Good for assembling many plots into one
- **ggplot2**
  - Grammar-based approach
  - Good for data exploration

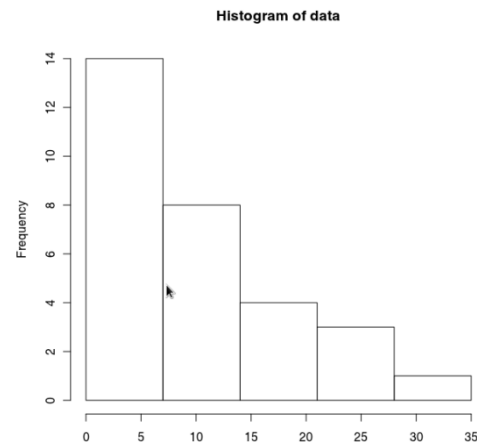
```
ggplot(d, aes(x=wt, y=mpg, colour=cyl))
+ geom_line()
+ geom_point(...)
+ geom_smooth(...)
+ facet_grid(model ~ make)
+ ...
```



- Illustrates frequencies of discrete intervals (bins)
- Used to plot density of data
- Bin size important
  - May hide data when using inappropriate bin sizes
- `data <- c(25, 11, 2, 13, 1, 22, 15, 3, 1, 7, 8, 10, 32, 7, 4, 9, 18, 21, 7, 7, 16, 6, 4, 12, 5, 27, 7, 9, 10, 7)`

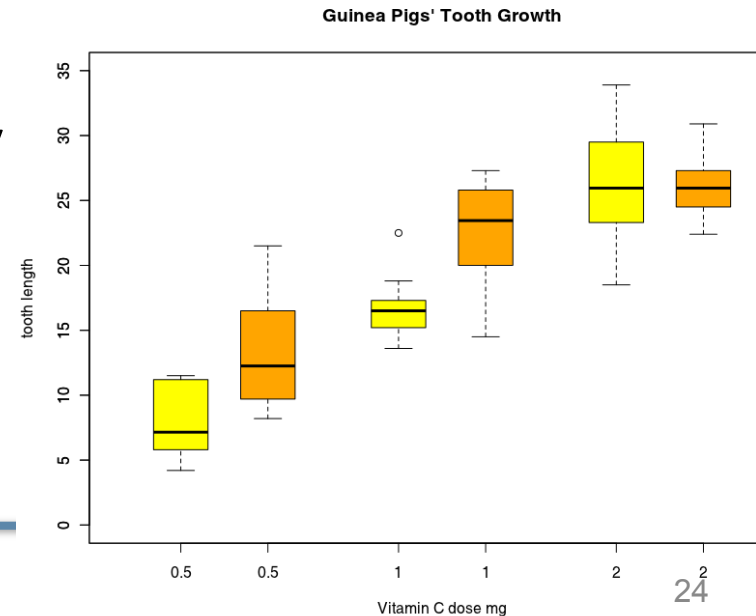


– `hist(data, breaks=c(0,6,13,20,27,34))`



`hist(data, breaks=c(0,7,14,21,28,35))`

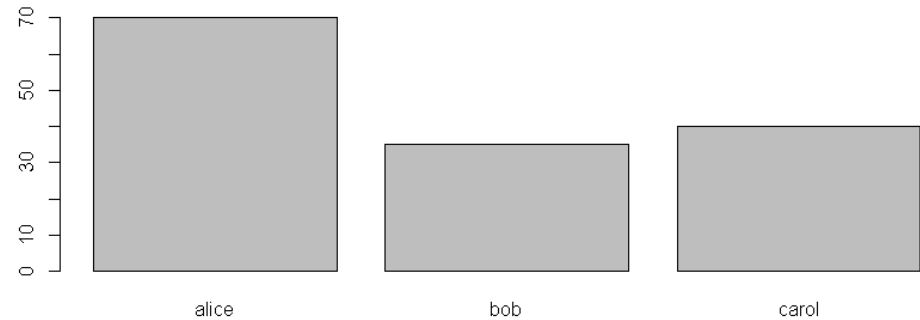
- Give a quick overview of the following values:
  - Min, 1<sup>st</sup> quantile, median, 3<sup>rd</sup> quantile, max
  - Whiskers:
    - extend to the most extreme data point which is no more than 1.5 times the interquartile range from the box
    - absolute minimum and maximum
- Outliers are displayed separately
  - if the minimum or maximum are not within the 1.5 IQR
- Advantages:
  - Allows to compare different result sets quickly by using multiple box plots
- Disadvantages:
  - Bimodal distributions cannot be spotted



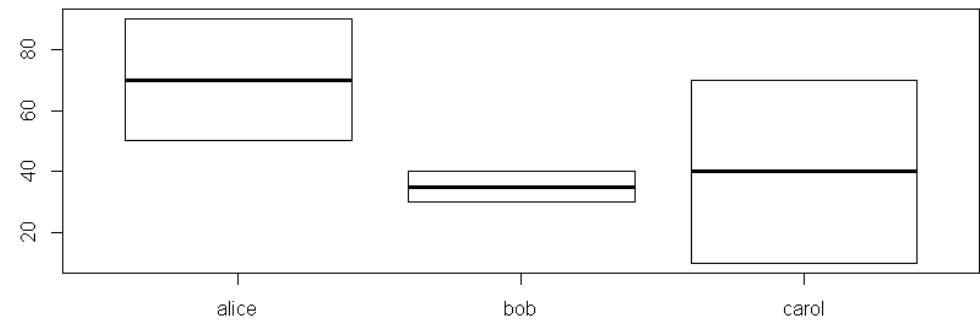




- Barplot
  - `barplot(a$x, names.arg=a$who)`

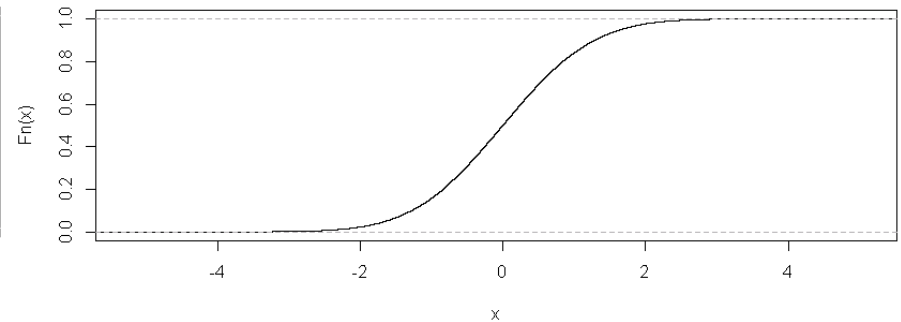
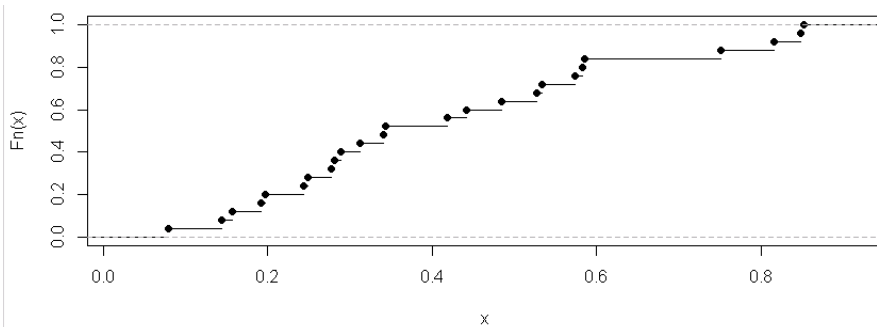


- Boxplot
  - `boxplot(d$points ~ d$student)`



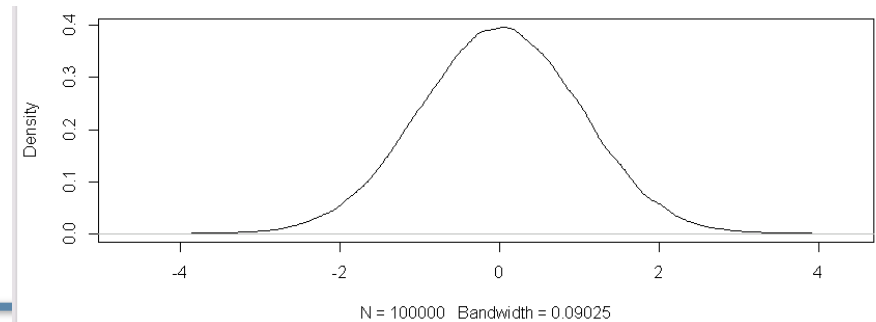
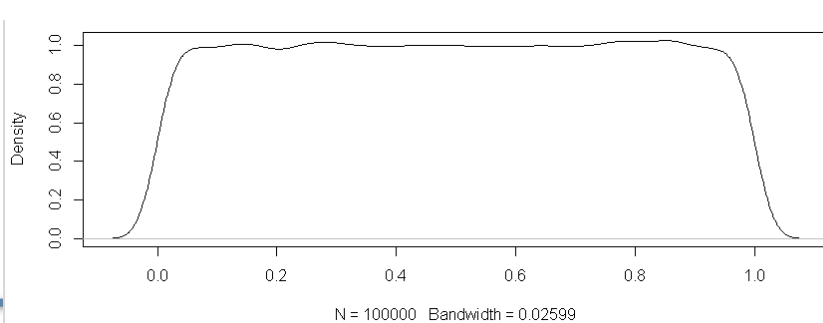
- eCDF

- `plot(ecdf(runif(25, min=0, max=1)))`
- `plot(ecdf(rnorm(1e5)))`



- Kernel Density Estimate

- `plot(density(runif(1e5, min=0, max=1)))`
- `plot(density(rnorm(1e5)))`

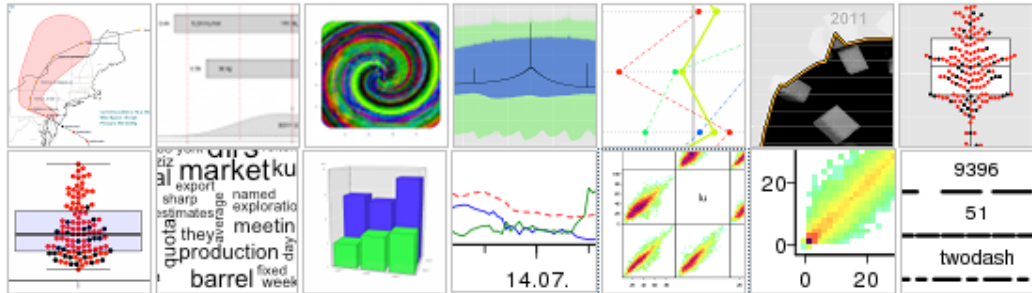




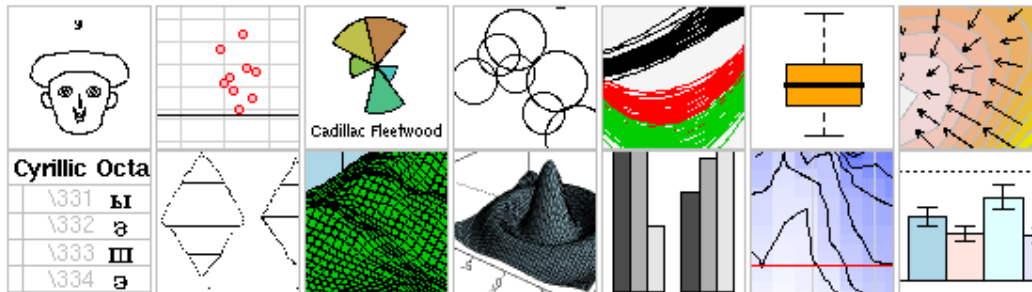
Donate



» Last entries ...



» Random entries



## R Project

R is a system for statistical computation and graphics. It consists of a language plus a run-time environment with graphics, a debugger, access to certain system functions, and the ability to run programs stored in script files.



## R Graphic Engine

One of R's strengths is the ease with which well-designed publication-quality plots can be produced, including mathematical symbols and formulae where needed. Great care has been taken over the defaults for the minor design choices in graphics, but the user retains full control.

Recent news from R-bloggers

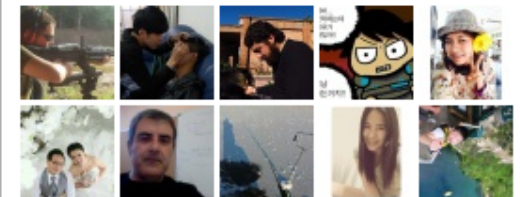
[Decluttering ordination plots in vegan part 1: ordilabel\(\)](#)

[The Cluster Bootstrap](#) [www.r-bloggers.com/](http://www.r-bloggers.com/)

Find us on Facebook

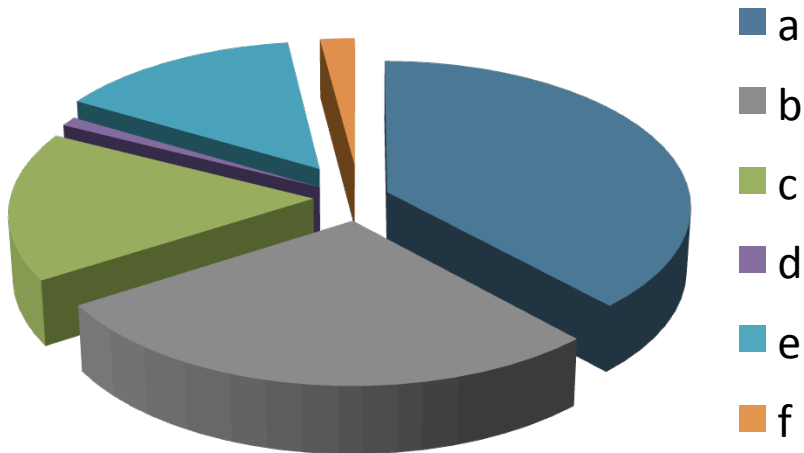


1,579 people like R Graph Gallery.



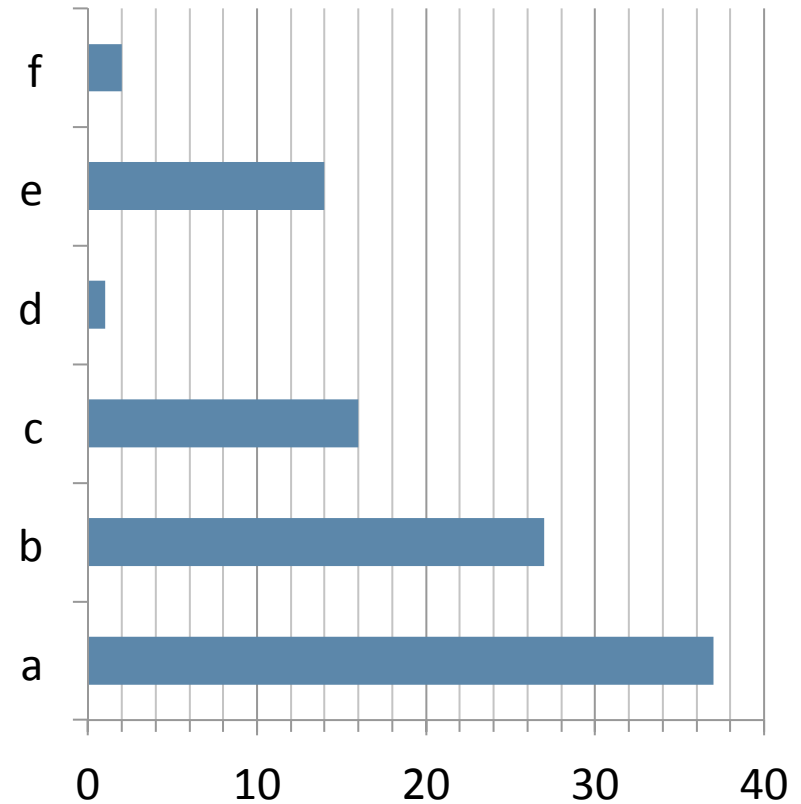
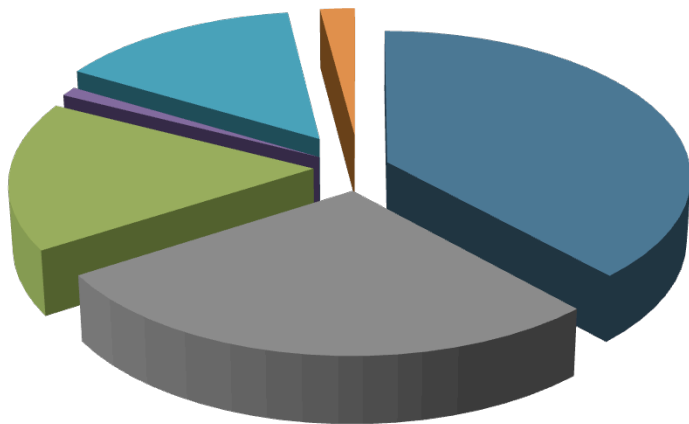
Facebook social plugin

- Pie charts are almost never a good idea
  - Humans cannot perceive area, angle or perimeter properly
  - Hence comparison of quantities is difficult





- Compare to bar charts:





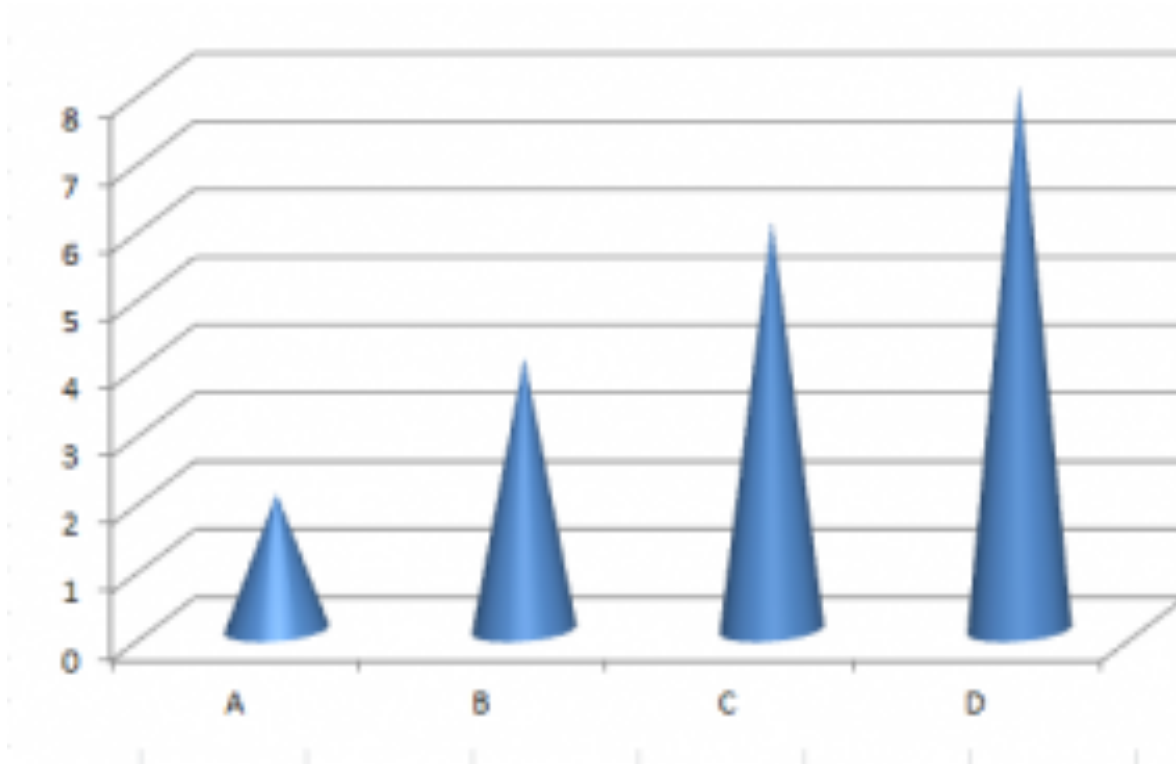
- Spot the differences!



<http://blog.revolutionanalytics.com/2009/08/how-pie-charts-fail.html>



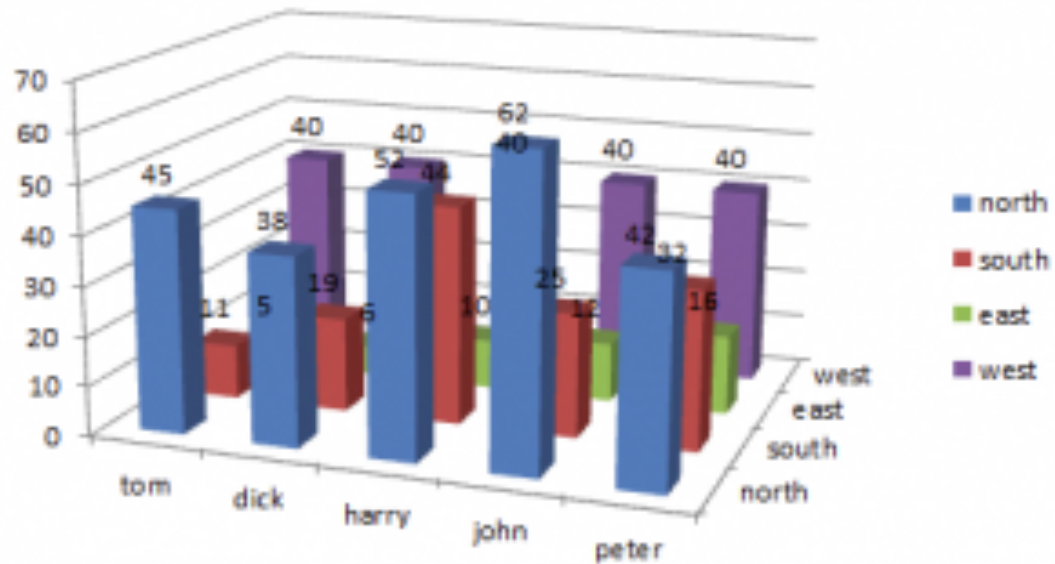
# Bad Graphs



<http://www.forbes.com/sites/naomirobbins/2012/05/30/winner-of-the-bad-graph-contest-announced-2/>



# Bad Graphs

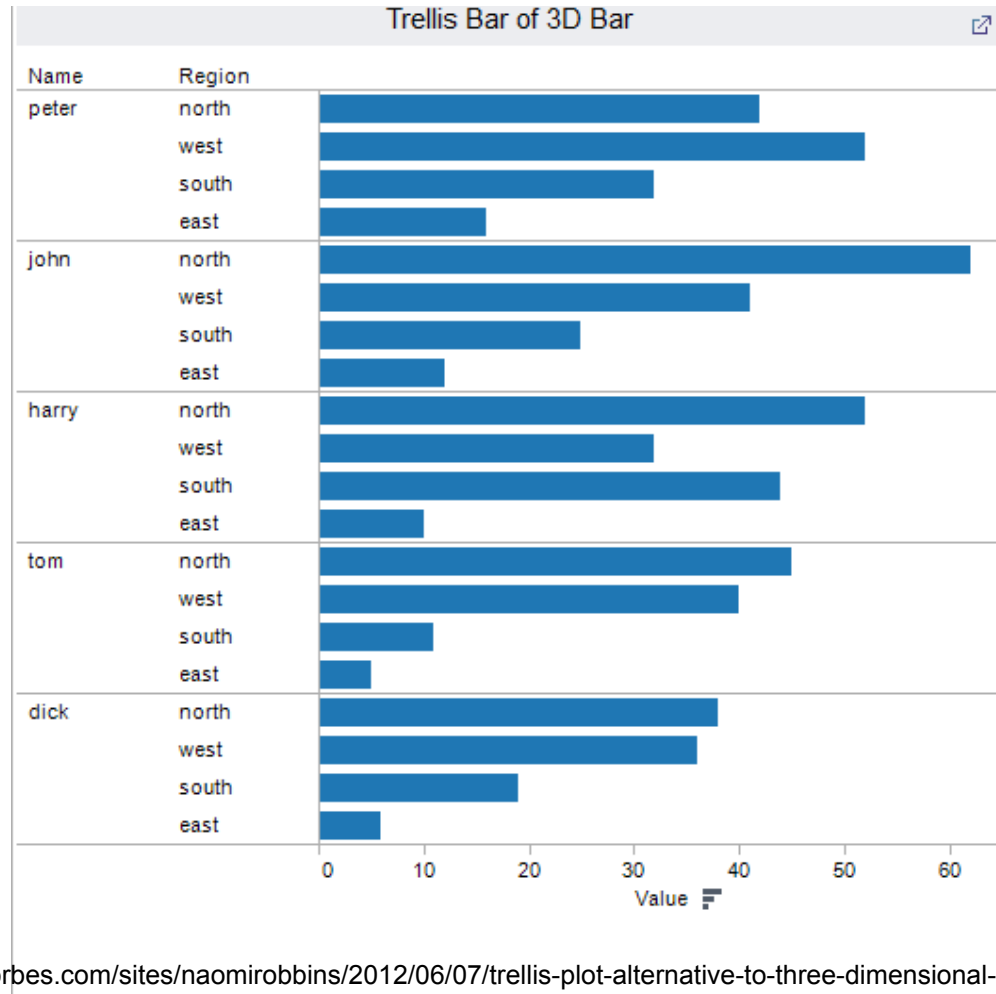


<http://www.forbes.com/sites/naomiobbins/2012/05/30/winner-of-the-bad-graph-contest-announced-2/>

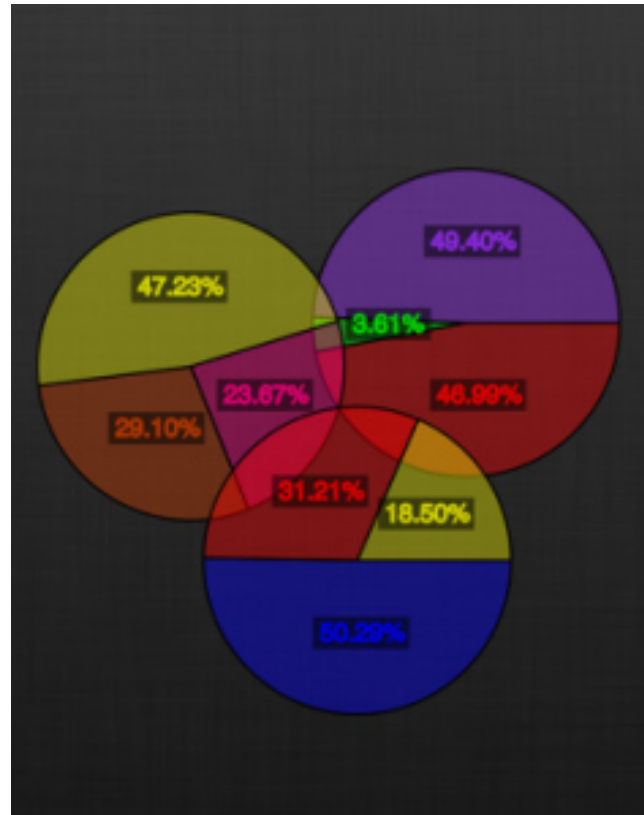




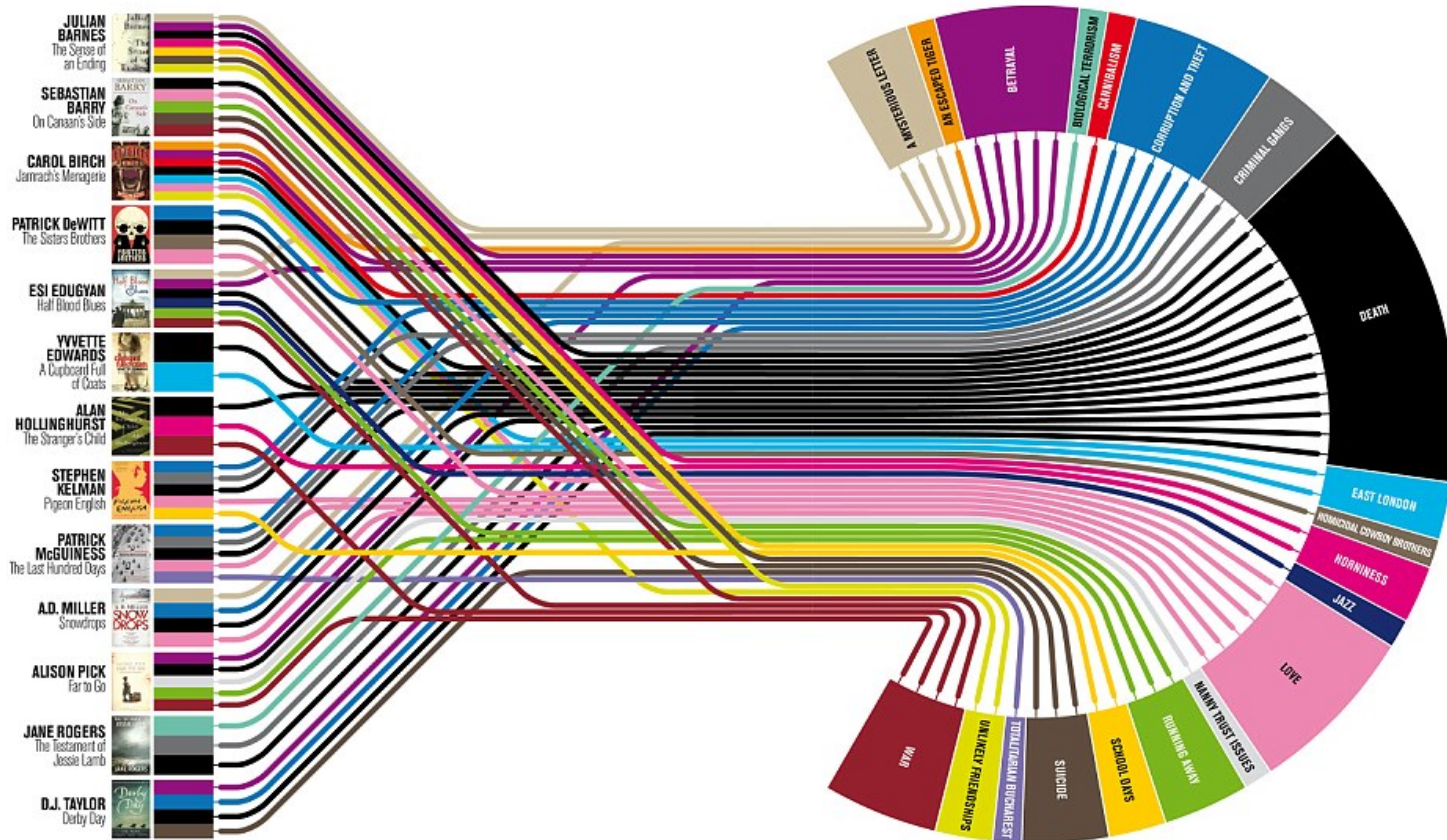
# Bad Graphs



<http://www.forbes.com/sites/naomirobbins/2012/06/07/trellis-plot-alternative-to-three-dimensional-bar-charts/>



<http://www.forbes.com/sites/naomirobbs/2012/05/30/winner-of-the-bad-graph-contest-announced-2/>



[http://junkcharts.typepad.com/junk\\_charts/2012/10/can-information-be-beautiful-when-information-doesnt-exist.html](http://junkcharts.typepad.com/junk_charts/2012/10/can-information-be-beautiful-when-information-doesnt-exist.html)



- Global Warming!!!

Average Monthly Temperature	
	Fahrenheit
January	26.4
February	28.0
March	35.8
April	46.6
May	56.3
June	65.8
July	71.2

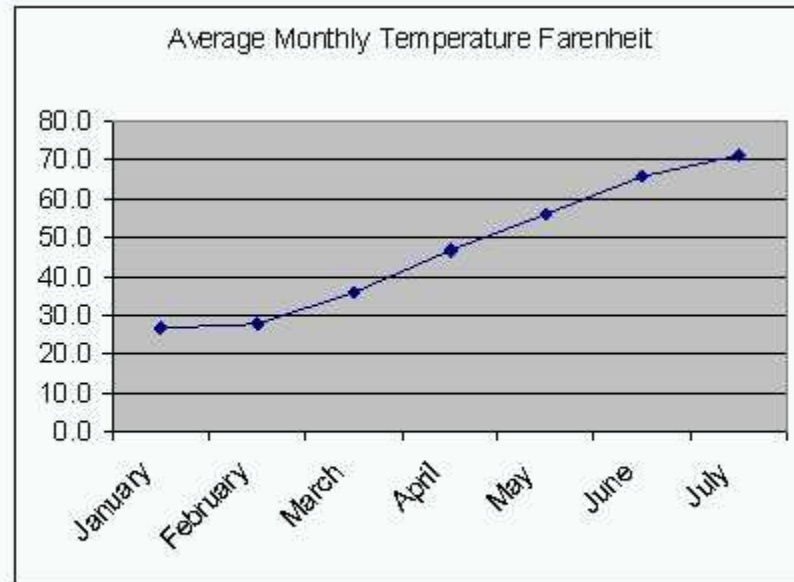
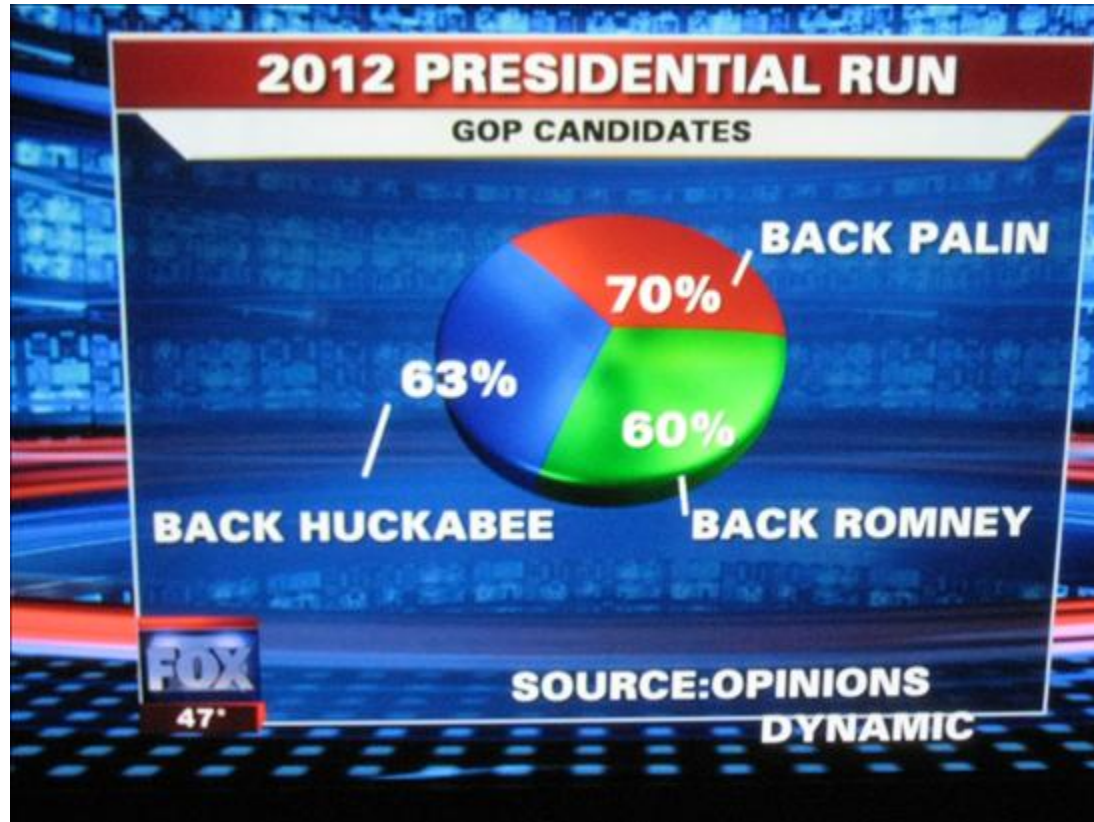


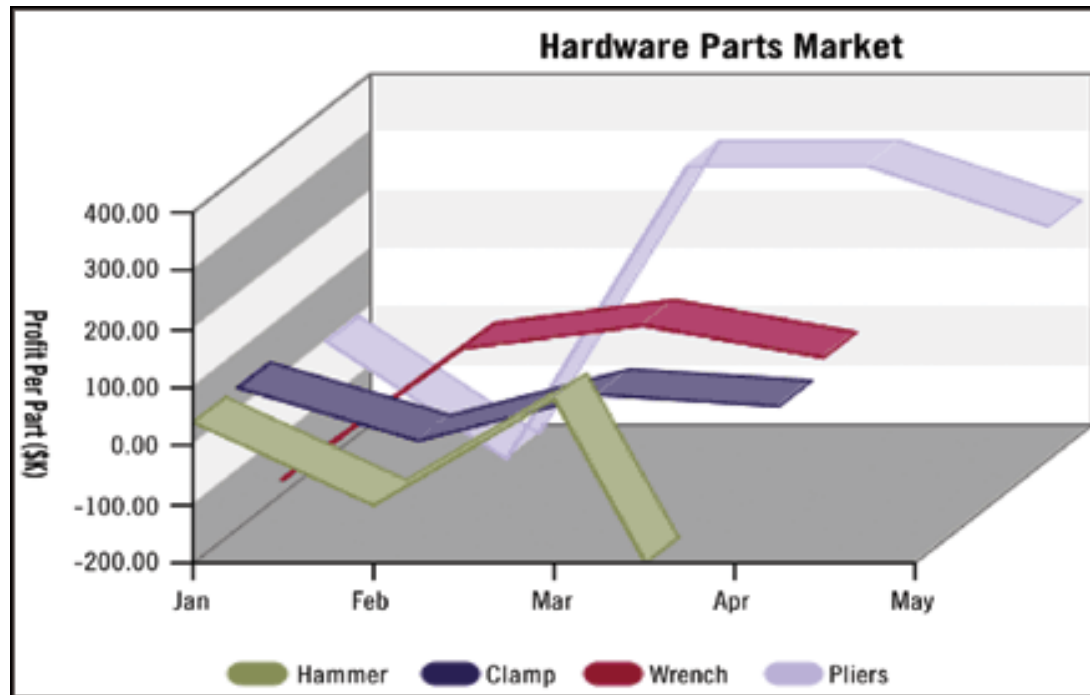
Figure 2

- For a total of... ???





# Bad Graphs







- With R you can quickly plot a graph: good
- Result is satisfying, how do I save it?
  - Open a graphic driver with `pdf()`, `png()`, `svg()`, `jpeg()`, ..., even LaTeX
  - `n <- rnorm(1000)`
  - `pdf('myplot.pdf')`
  - `plot(n, type='l')`
  - `dev.off()`



- **CHECKLIST FOR PLOTS BEFORE HANDING IN THE ASSIGNMENT**
- Make plots readable by
  - Not showing too much data, plot should be readable in seconds!
  - Using the “right” plot for visualization purpose
  - Do not rely on **COLOR** alone
    - plot \*will\* be printed out in greyscale, or black-and-white
  - Make it easy to compare the quantities
  - **AXIS**: what’s on abscissa? what’s on ordinate?
  - **UNITS**: without units values makes no sense
    - meters? millimeters? light years? bananas? potatoes?
  - Check your plot in the final document. **PRINT IT OUT**
    - a document on your screen might not be as good in a double column document
  - Use **PDF** as output format!!! **NO HORRIBLE RASTER GRAPHICS!!!**





- Perform some simple data analysis / plotting with R
  - download and install R
  - download and install a front end (R Studio might be the the quickest option)
  - download the meteo dataset from classroom (taken from [www.soda-is.com](http://www.soda-is.com))
- First steps
  - load the dataset in R (function `read.csv()`) (see next slide for field description)
  - what is inside the dataset? (function `names()`, or `head()`)
  - which cities are included? (function `unique()`)
  - can you get min, max, quartiles, and average for the average temperatures? (function `summary()`)
  - can you split the previous stats by city? (`unique()`, `for loop`, `subset()`)
    - for the braves: install the `plyr` package and to this with `ddply()`



- Month: Value for each month.
- City: Guess what?
- IrradianceWm2: Monthly mean of irradiance (W/m<sup>2</sup>)
- IrradianceKLyYear: Monthly mean of daily irradiance (kLy/year)
- IrradiationJcm2: Monthly mean of daily irradiation (J/cm<sup>2</sup>)
- TempMin: Monthly mean of daily min. of air temperature (C)
- TempMax: Monthly mean of daily max. of air temperature (C)
- TempMean: Monthly mean of air temperature (C)
- RelHumMin: Monthly mean of daily min. of relative humidity (%)
- RelHumMax: Monthly mean of daily max. of relative humidity (%)
- RelHumMean: Monthly mean of relative humidity (%)



- Make some plots:
  - mean temperature trend for Trento in a year (high level plot commands)
  - mean temperature trend for all cities in a year (one line each: use low level plot commands)
  - add a new column to the data.frame labeling the month (JAN, FEB, MAR...). Can you do it in a smart way? Then replot the previous graph with the name instead of the number
  - boxplots of mean temperature for different cities (look at the documentation for `boxplot()`, parameter “formula”). What can you infer from this plot?
  - add a column for the season (again, smart way) and plot the boxplots of mean temperatures of Trento for the different seasons
  - use `ddply` to compute the total irradiation for a city in a year ( $\text{W}/\text{m}^2$ ) and make a barplot of such values. Would you go selling solar panels in Eureka? (assume each month has 30 days)
  - bonus: use low level plotting functions for the previous plot and make the y axis nicer ( $\text{kWh}/\text{cm}^2$ )



## Bonus! System Analysis

- Problem
  - Your virtual machine performance are terrible
  - You want to understand where the problem might be
- How:
  - You want to analyze the performance of your virtual machine
  - Indicators: CPU utilization, RAM, swap, ...
  - Every 5 minutes, the indicators are logged into a csv file
  - By taking this dataset and analyzing its content, we'll try to get to a conclusion

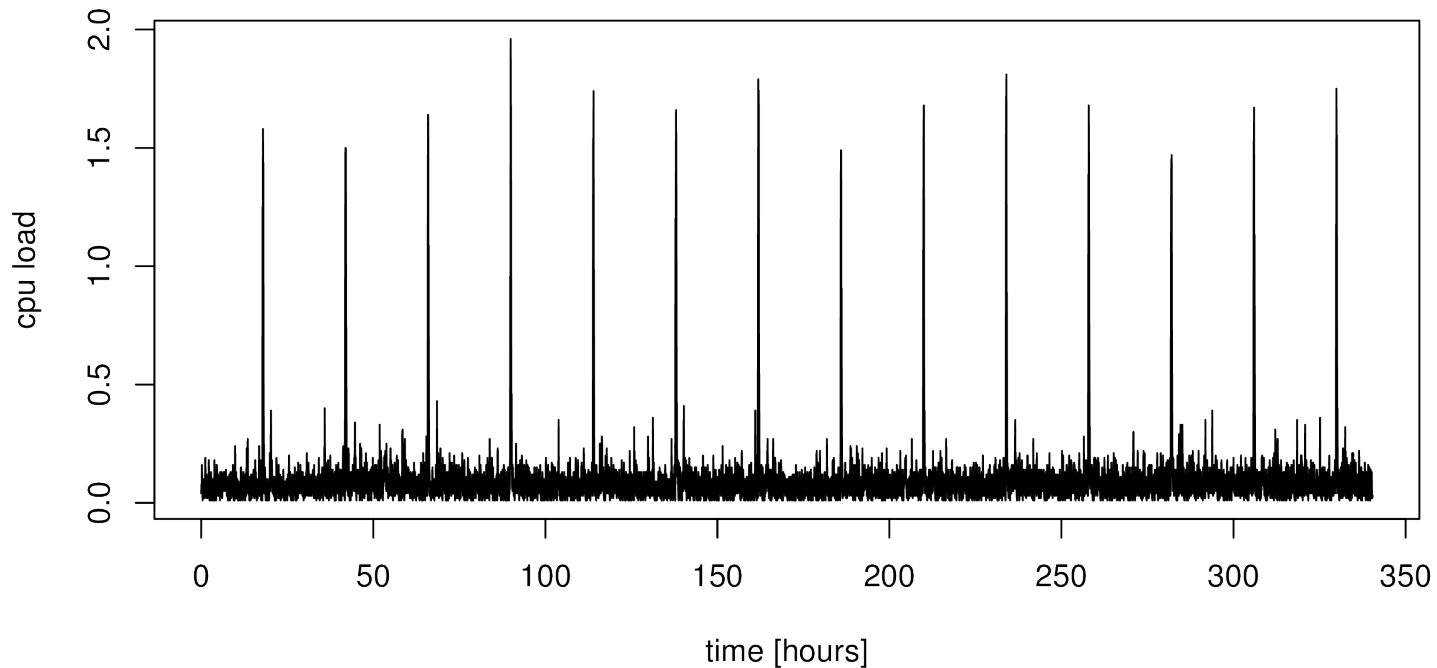


- Load your data into R:

```
d <- read.csv("measures.csv")
print(names(d))
[1] "timestamp" "tcpconnections" "swapload" "ramload" "rtodisk" "cpuload"
d$hours <- (d$timestamp - min(d$timestamp)) / 3600
```

- Let's see if the CPU is overloaded:

```
pdf('cpuload.pdf', width=16/2, height=9/2)
plot(d$hours, d$cpuload, type='l', xlab="time [hours]", ylab="cpu load")
dev.off()
```





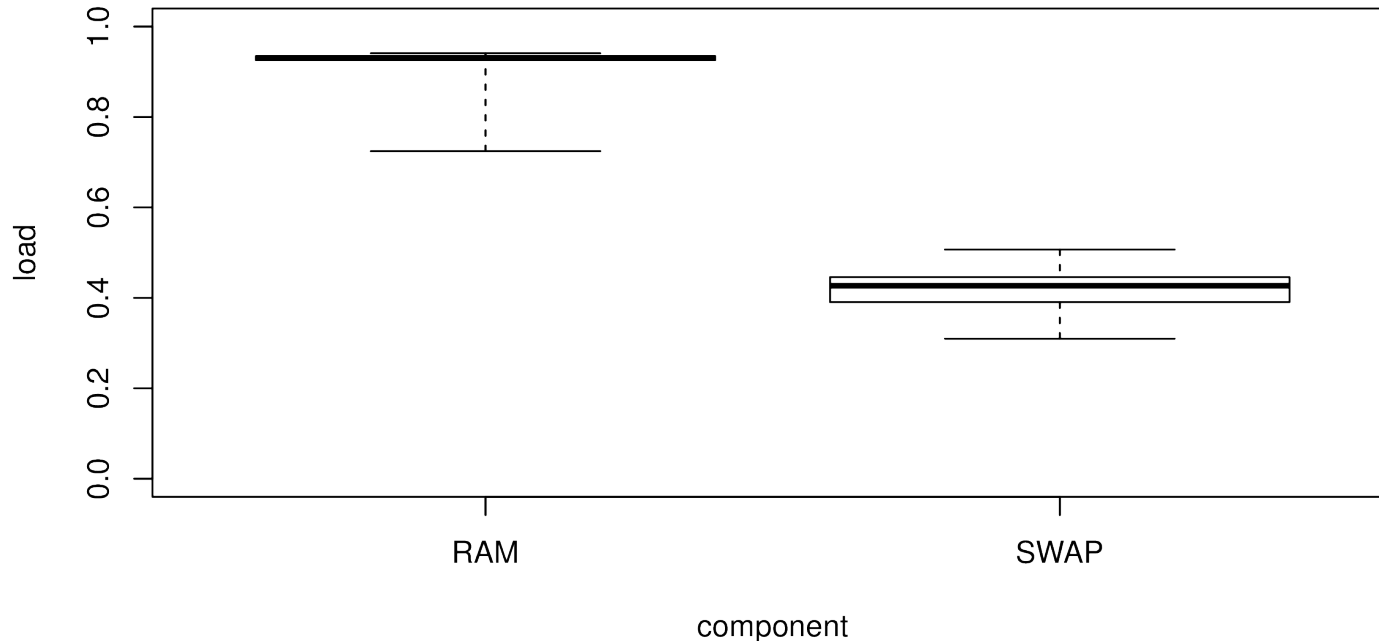
- When do the CPU load peaks happen?
  - `peaks <- d[ d$cpuload>0.5, ]$timestamp`
  - `print(as.POSIXlt(peaks, origin="1970-01-01"))`

```
[1] "2015-03-31 04:05:01 CEST" "2015-03-31 04:10:01 CEST" "2015-03-31 04:15:01 CEST" "2015-03-31 04:20:01 CEST" "2015-04-01 04:05:02 CEST"  
[6] "2015-04-01 04:10:01 CEST" "2015-04-01 04:15:01 CEST" "2015-04-01 04:20:01 CEST" "2015-04-02 04:05:01 CEST" "2015-04-02 04:10:02 CEST"  
[11] "2015-04-02 04:15:01 CEST" "2015-04-02 04:20:01 CEST" "2015-04-03 04:05:01 CEST" "2015-04-03 04:10:02 CEST" "2015-04-03 04:15:01 CEST"  
[16] "2015-04-03 04:20:01 CEST" "2015-04-04 04:05:01 CEST" "2015-04-04 04:10:01 CEST" "2015-04-04 04:15:01 CEST" "2015-04-04 04:20:01 CEST"  
[21] "2015-04-05 04:05:02 CEST" "2015-04-05 04:10:01 CEST" "2015-04-05 04:15:01 CEST" "2015-04-05 04:20:01 CEST" "2015-04-05 04:25:01 CEST"  
[26] "2015-04-06 04:05:01 CEST" "2015-04-06 04:10:01 CEST" "2015-04-06 04:15:01 CEST" "2015-04-06 04:20:01 CEST" "2015-04-07 04:05:01 CEST"  
[31] "2015-04-07 04:10:02 CEST" "2015-04-07 04:15:01 CEST" "2015-04-07 04:20:01 CEST" "2015-04-08 04:05:01 CEST" "2015-04-08 04:10:01 CEST"  
[36] "2015-04-08 04:15:01 CEST" "2015-04-08 04:20:01 CEST" "2015-04-09 04:05:02 CEST" "2015-04-09 04:10:01 CEST" "2015-04-09 04:15:01 CEST"  
[41] "2015-04-09 04:20:02 CEST" "2015-04-10 04:05:02 CEST" "2015-04-10 04:10:01 CEST" "2015-04-10 04:15:02 CEST" "2015-04-10 04:20:01 CEST"  
[46] "2015-04-11 04:05:01 CEST" "2015-04-11 04:10:01 CEST" "2015-04-11 04:15:01 CEST" "2015-04-11 04:20:01 CEST" "2015-04-12 04:05:01 CEST"  
[51] "2015-04-12 04:10:02 CEST" "2015-04-12 04:15:01 CEST" "2015-04-12 04:20:01 CEST" "2015-04-12 04:25:01 CEST" "2015-04-13 04:05:01 CEST"  
[56] "2015-04-13 04:10:01 CEST" "2015-04-13 04:15:01 CEST" "2015-04-13 04:20:01 CEST"
```

- CPU load due to backups

- Is it a memory problem?

```
pdf('memload.pdf', width=16/2, height=9/2)
boxplot(list("RAM"=d$ramload, "SWAP" = d$swapload), range=0,
        ylim=c(0, 1), xlab="component", ylab="load")
dev.off()
```



- Is this because we don't have enough memory?





- Is there a problem with the webserver?

```
time.plot <- function(x, y, col, xlim, ylim, xlab, ylab) {  
  plot.new()  
  plot.window(xlim=xlim, ylim=ylim)  
  
  lines(x, y, col=col)  
  axis(1)  
  axis(2)  
  title(xlab=xlab, ylab=ylab, line=2)  
  box()  
}  
  
pdf('tcpload.pdf', width=16/2, height=9/2)  
old <- par(mfrow=c(3,1), mai = c(0.5, 0.5, 0.1, 0.3))  
xlim <- c(min(d$hours), max(d$hours))  
time.plot(d$hours, d$tcpconnections, 'black', xlim, ylim=c(0, 60), xlab="time [hours]", ylab="tcp connections")  
time.plot(d$hours, d$ramload, 'red', xlim, ylim=c(0, 1), xlab="time [hours]", ylab="ram load")  
time.plot(d$hours, d$swapload, 'blue', xlim, ylim=c(0, 1), xlab="time [hours]", ylab="swap load")  
par(old)  
dev.off()
```



# Steps

