# Data visualisation with Ruby

Chris Lowis (@chrislowis)
BBC A&M&M

# The Problem:

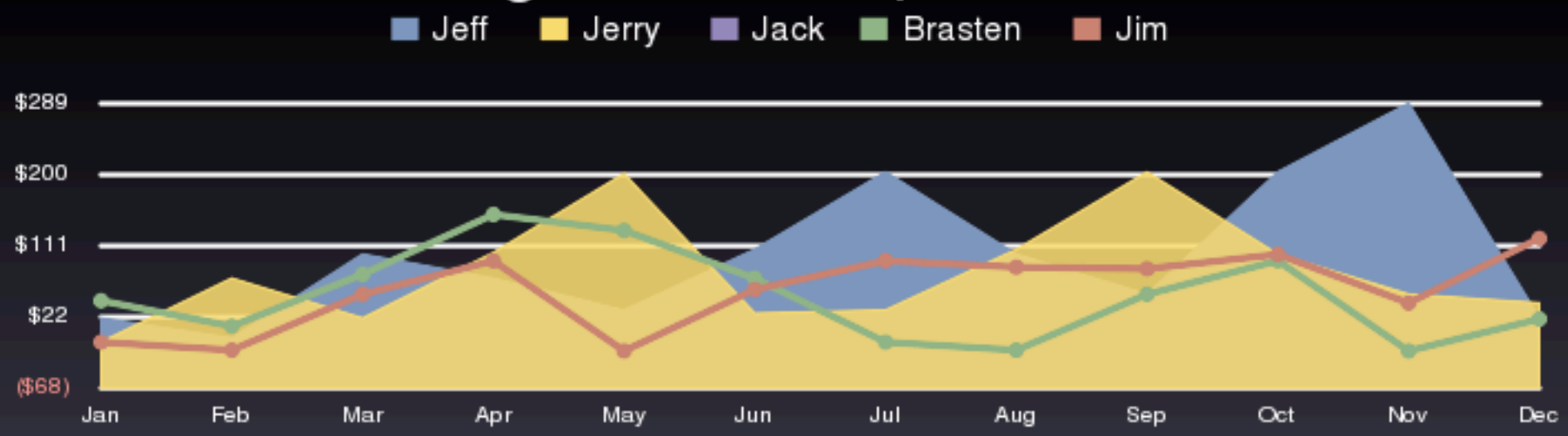
# We have some data

To display to our users
or ...

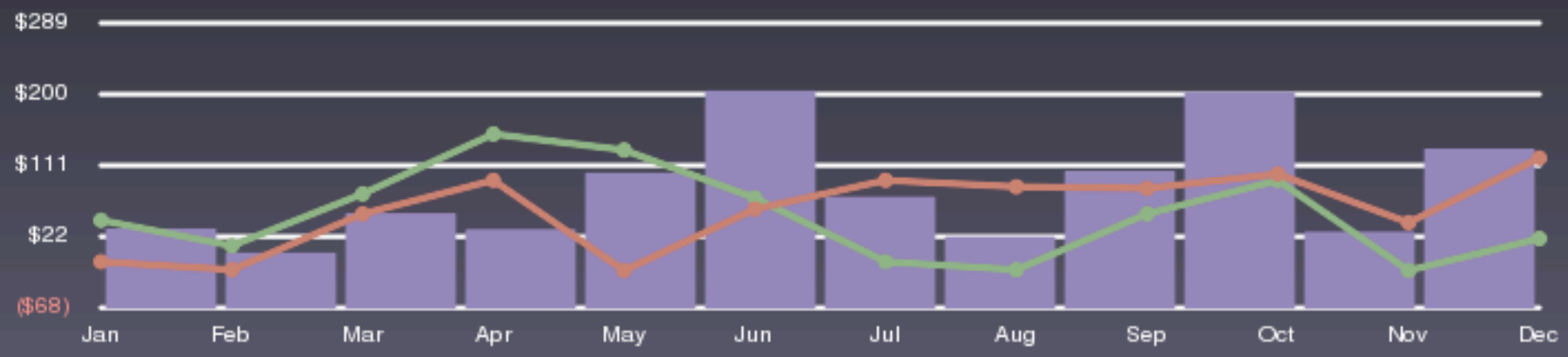
... to explore and understand.

There are few native data visualisation tools for Ruby

# Long-term Comparisons




Northeastern (Top) / Central (Bottom)



# Scruffy

- SVG based graphs
- Basic, functional plotting
- Not a lot of development activity recently
- Limited data analysis

# This talk

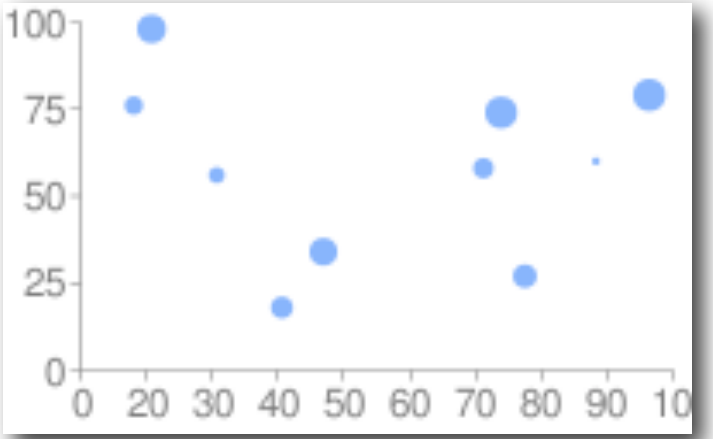
- A survey of alternative data visualisation libraries
- Basic examples with installation instructions for you to try at home.

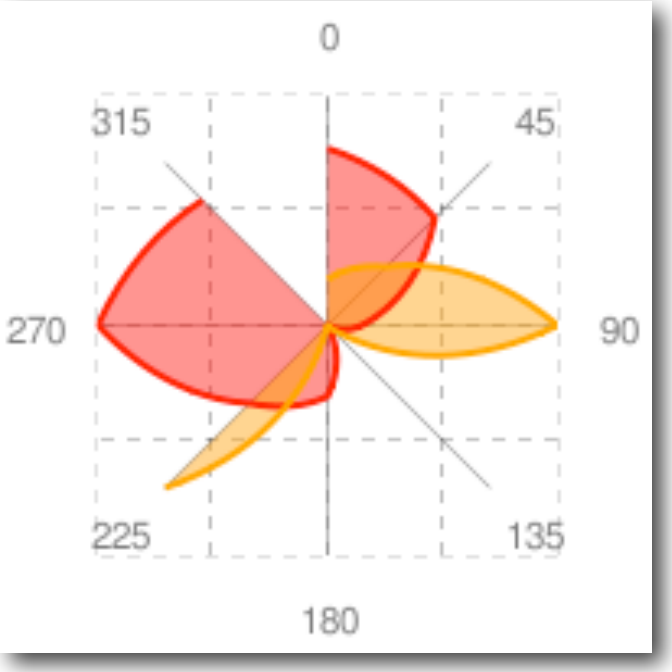
http://github.com/chrislo/data_visualisation_ruby

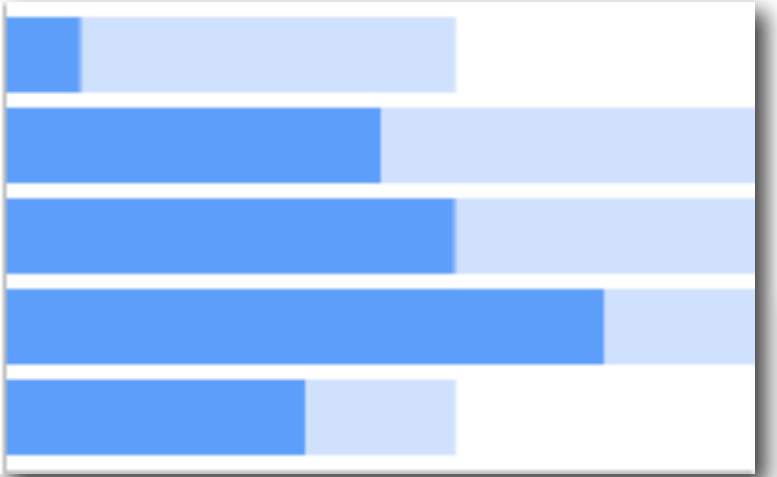
GOOGLE

```ruby
require 'rubygems'
require 'google_chart'
require File.join(File.dirname(__FILE__),'..','stock_data.rb')

x = []; y = []
data = StockData.for('RBS.L', 365)
data.each {|d| x << d[0]; y << d[1]}
y.reverse!

# Line Chart
GoogleChart::LineChart.new('400x300', "RBS.L", false) do |lc|
  lc.data "RBS.L", y, '0000ff'
  lc.grid(:x_step => 100.0/12.0,
          :y_step => 100.0/6.0,
          :length_segment => 1,
          :length_blank => 0)
  lc.axis(:x,
          :range => [0,12],
          :labels => %w(d j f m a m j j a s o n d)
          )
  puts lc.to_url
end
```


RBS.L
d
j
f
m
a
m
j
j
a
s
o
n
d
RBS.L


100
75
50
25
0
0 20 30 40 50 60 70 80 90 10


0
315
45
270
90
225
135
180



# TIMETRIC

- Data analysis webservice
- Upload your data through the site, or the API
- Compare your data with public datasets, share, embed or basic analysis.

```ruby
require 'base64'
require 'net/http'
require '../stock_data.rb'
require 'yaml'

# You need an API key from timetric.com. I've put mine in a yaml file and
# loaded it from there.
authentication = YAML.load_file(ENV['HOME'] + '/.timetric_api')
key = authentication[:key]
secret = authentication[:secret]
basic_str = "Basic #{Base64.b64encode(key+":"+secret)}"

# Use our simple stock market class to get some time-history data to
# play with.
data = StockData.csv_for('TSCO', 365)

# Make a post request to timetric to upload the time series.
http = Net::HTTP.new('timetric.com')

headers = {
   'Authorization' => basic_str,
   'Content-Type' => 'text/csv'
}
res = http.post('/create/?title=tsco&caption=prices', data, headers)
puts res.body
```

timetric

Dashboard | Settings | Help | Log out

# Dashboard

Search

From here you can view your starred series, manage the series you've created, and build new ones using our formula language.

Create New Series

- Starred
- My Series
- Recently Viewed

Build | Overlay | Versus | Bar Chart

| | | |
|---|---|---|
| ☐ | test | 2.32 |
| ☐ | rbsl | 33 |
| ☑ | rbsl | 33 |
| ☐ | 10-point moving mean over 'rbsl' | 34.62 |
| ☐ | rbsl | 50.43 |
| ☐ | 10-point moving mean over 'Replies to @twitter on Twitter' | |
| ☑ | tsco | 50.43 |

About Timetric | Help | FAQs | Community Support | Timetric's API | Contact us | © Inkling Software, 2009

timetric

Register Now! | Help | Log in

Overlay:

rbsl and tsco

Multiple Axes

Share this graph

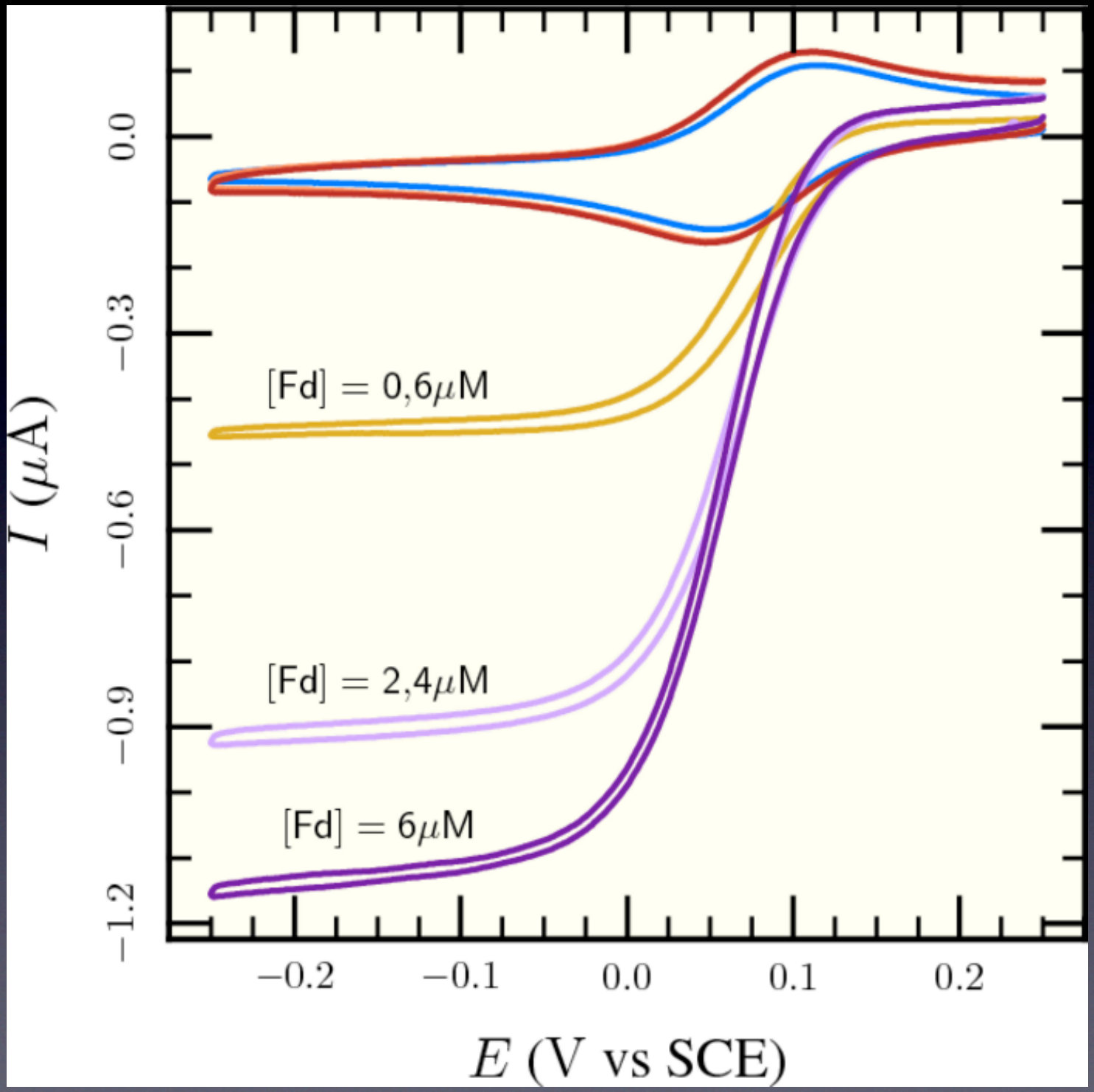

## Other Visualizations

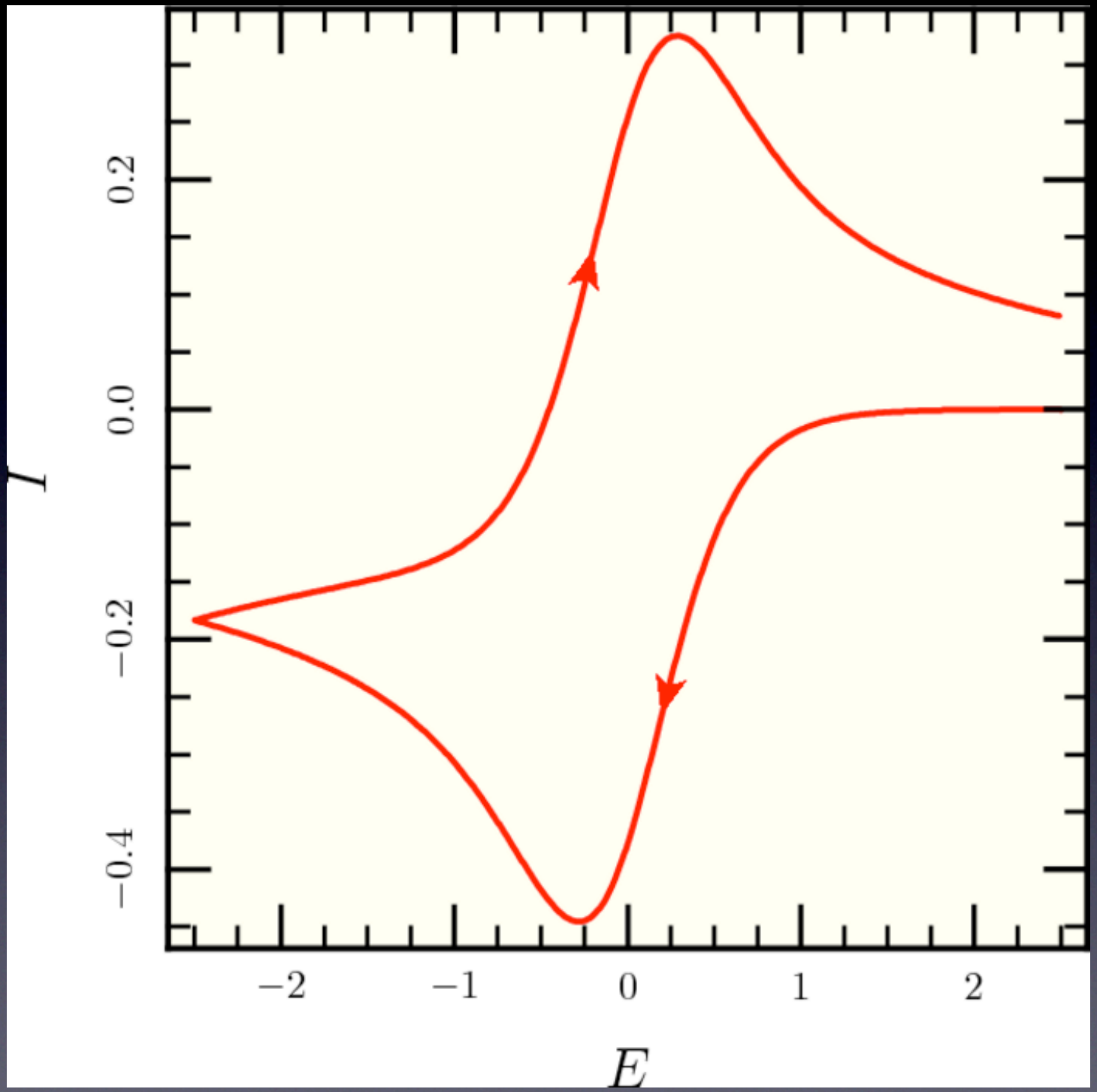
- Versus graph
- Bar chart
- Sharable List

About Timetric | Help | FAQs | Community Support | Timetric's API | Contact us

# TIOGA

- Uses LaTeX as its backend (great for mathematics, PDF output)
- No data analysis.

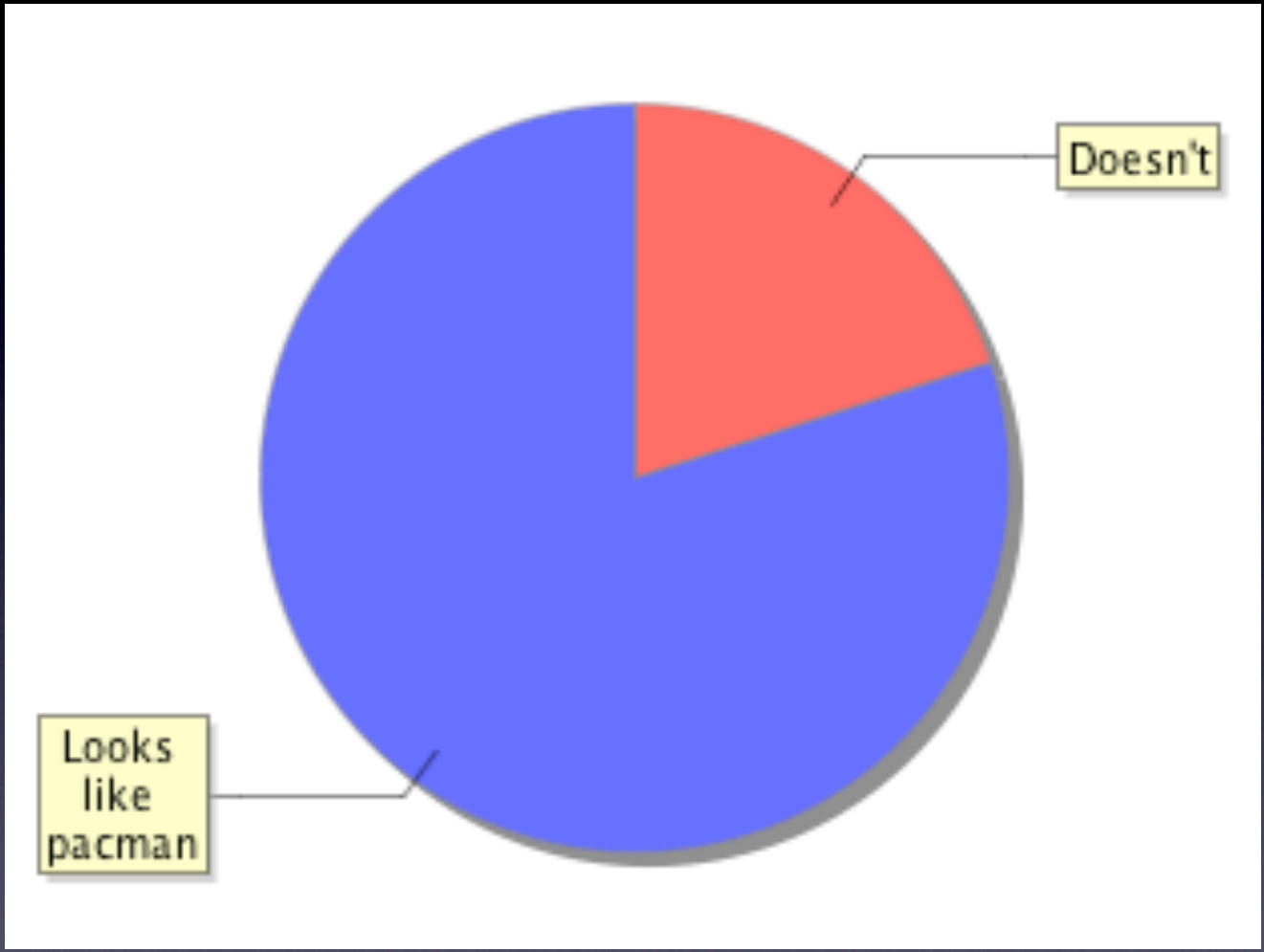

0.0
−0.3
−0.6
−0.9
−1.2
I (μA)
[Fd] = 0,6μM
[Fd] = 2,4μM
[Fd] = 6μM
−0.2
−0.1
0.0
0.1
0.2
E (V vs SCE)

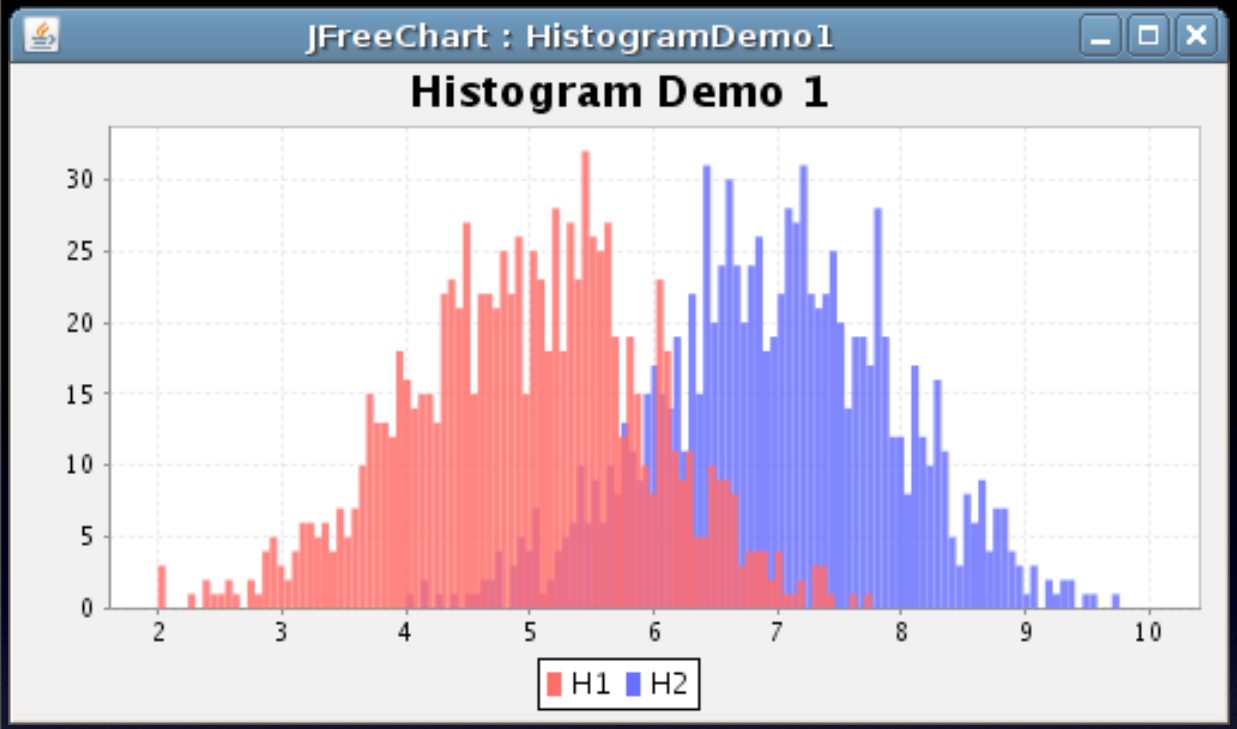

0.2
0.0
−0.2
−0.4
I
−2
−1
0
1
2
E

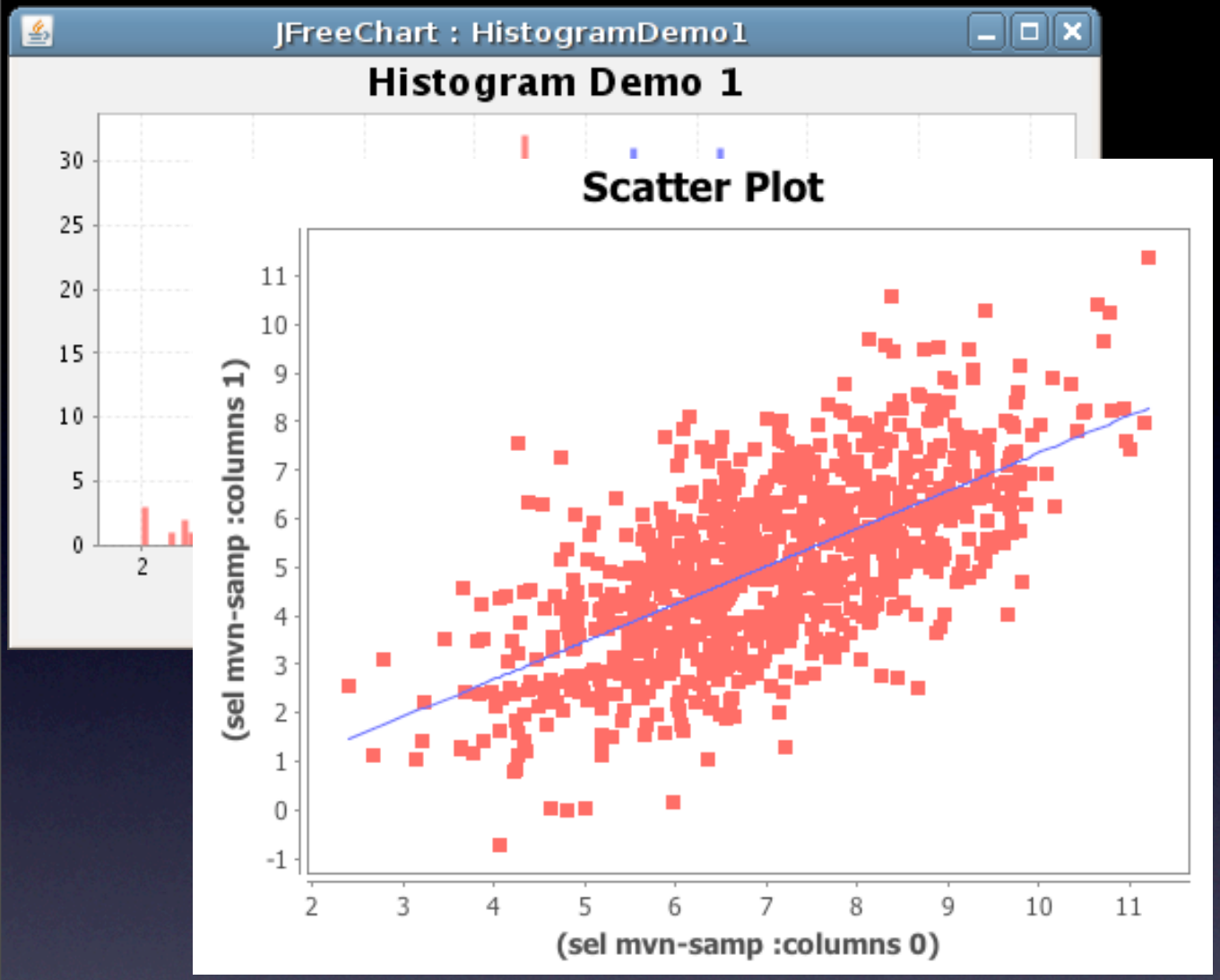
# JFREECHART

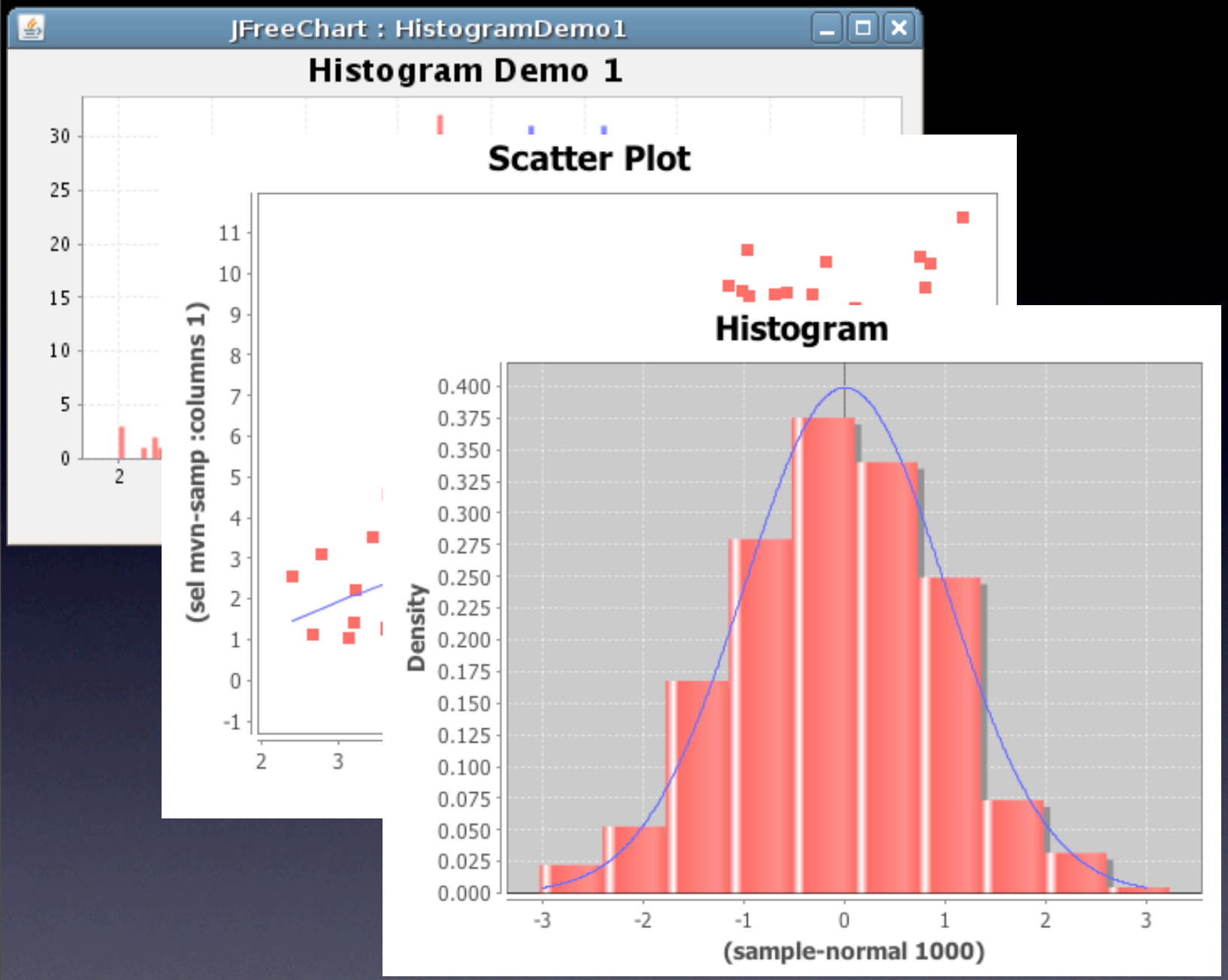
```ruby
class Plot
  require 'java'
  require 'jcommon-1.0.16'
  require 'jfreechart-1.0.13'

  include_class 'java.io.File'
  include_class 'org.jfree.chart.ChartUtilities'
  include_class 'org.jfree.chart.JFreeChart'
  include_class 'org.jfree.chart.plot.PiePlot'
  include_class 'org.jfree.data.general.DefaultPieDataset'

  def initialize(width=400, height=600)
    @width = width
    @height = height
    dataset = pie_data
    @chart = create_chart(dataset)
  end

  def render_to_file(filename, format="png")
    javafile = java.io.File.new(filename)
    ChartUtilities.saveChartAsPNG(javafile, @chart, @width, @height)
  end
```

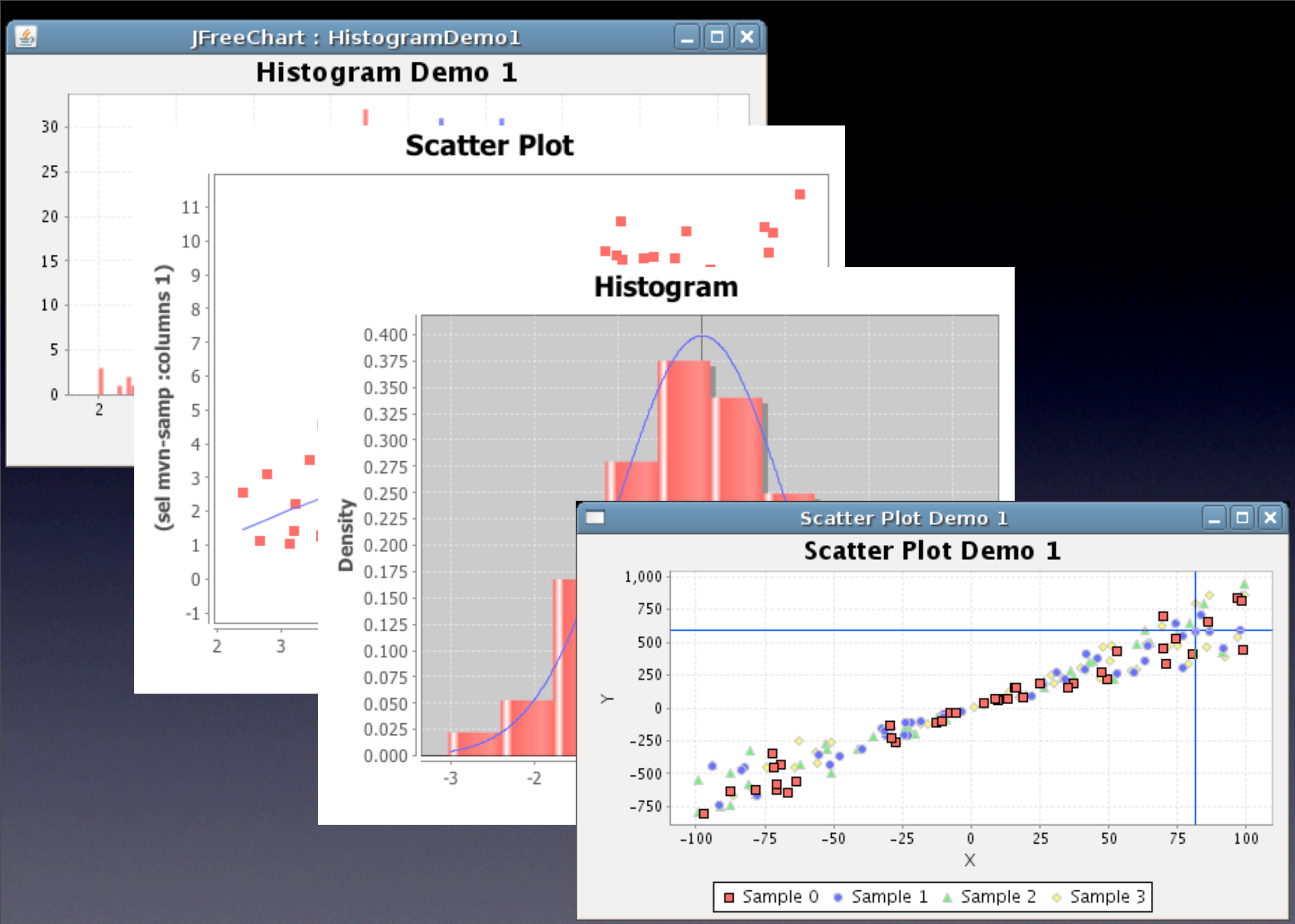
```
  private
  def pie_data
    dataset = DefaultPieDataset.new
    dataset.setValue("Doesn't", 20)
    dataset.setValue("Looks like pacman", 80)
    return dataset
  end

  def create_chart(dataset)
    plot = PiePlot.new
    plot.setDataset(dataset)

    chart = JFreeChart.new(nil, JFreeChart::DEFAULT_TITLE_FONT, plot, false)
    chart.setBorderVisible(false);
    return chart
  end
end

plot = Plot.new
puts "Rendering plot"
plot.render_to_file("plot.png")
```


Doesn't
Looks like pacman


JFreeChart : HistogramDemo1
Histogram Demo 1
30
25
20
15
10
5
0
2
3
4
5
6
7
8
9
10
H1
H2

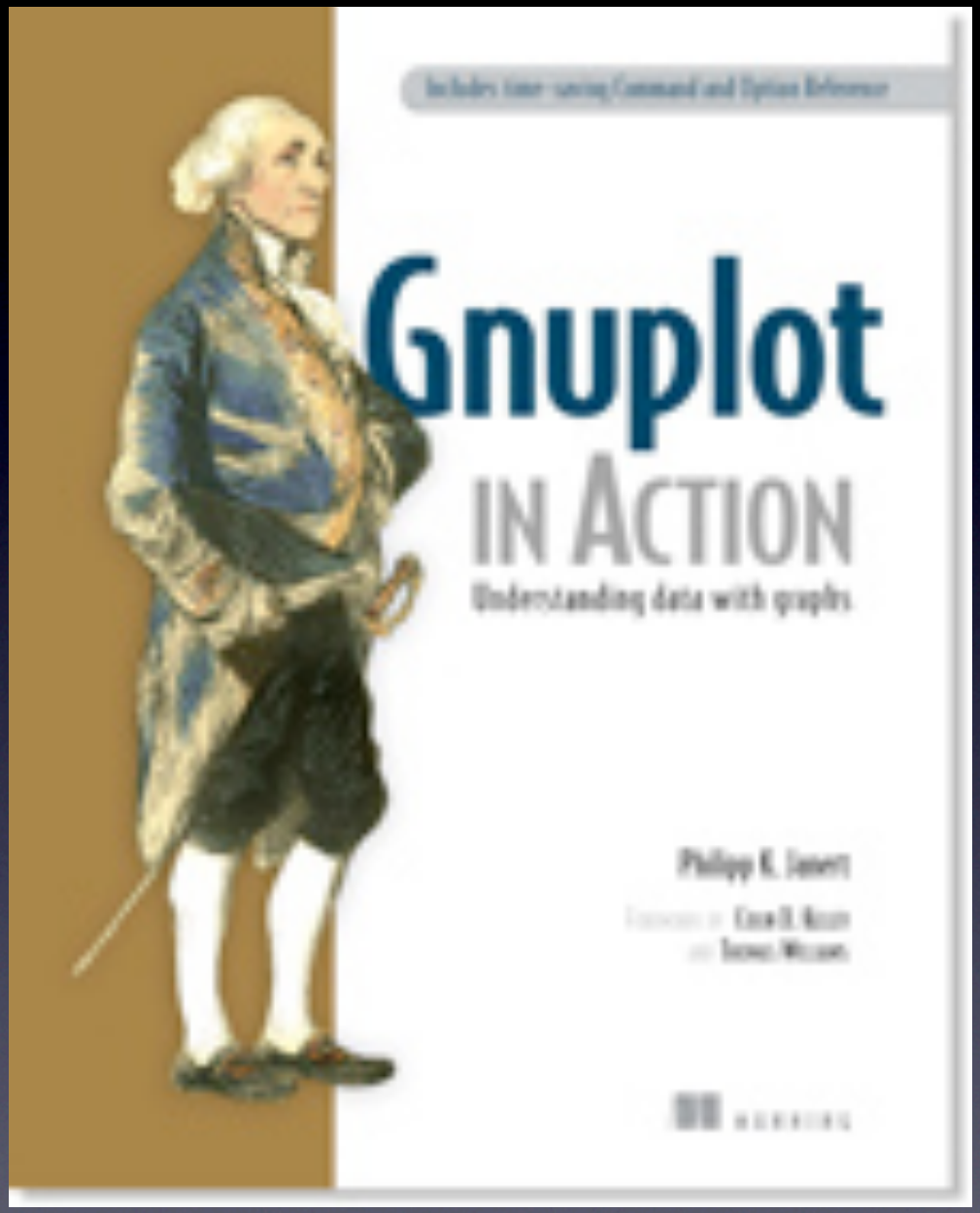

JFreeChart : HistogramDemo1
Histogram Demo 1
30
25
20
15
10
5
0
2
Scatter Plot
(sel mvn-samp :columns 1)
11
10
9
8
7
6
5
4
3
2
1
0
-1
2
3
4
5
6
7
8
9
10
11
(sel mvn-samp :columns 0)


JFreeChart : HistogramDemo1
Histogram Demo 1
Scatter Plot
(sel mvn-samp :columns 1)
Histogram
Density
(sample-normal 1000)

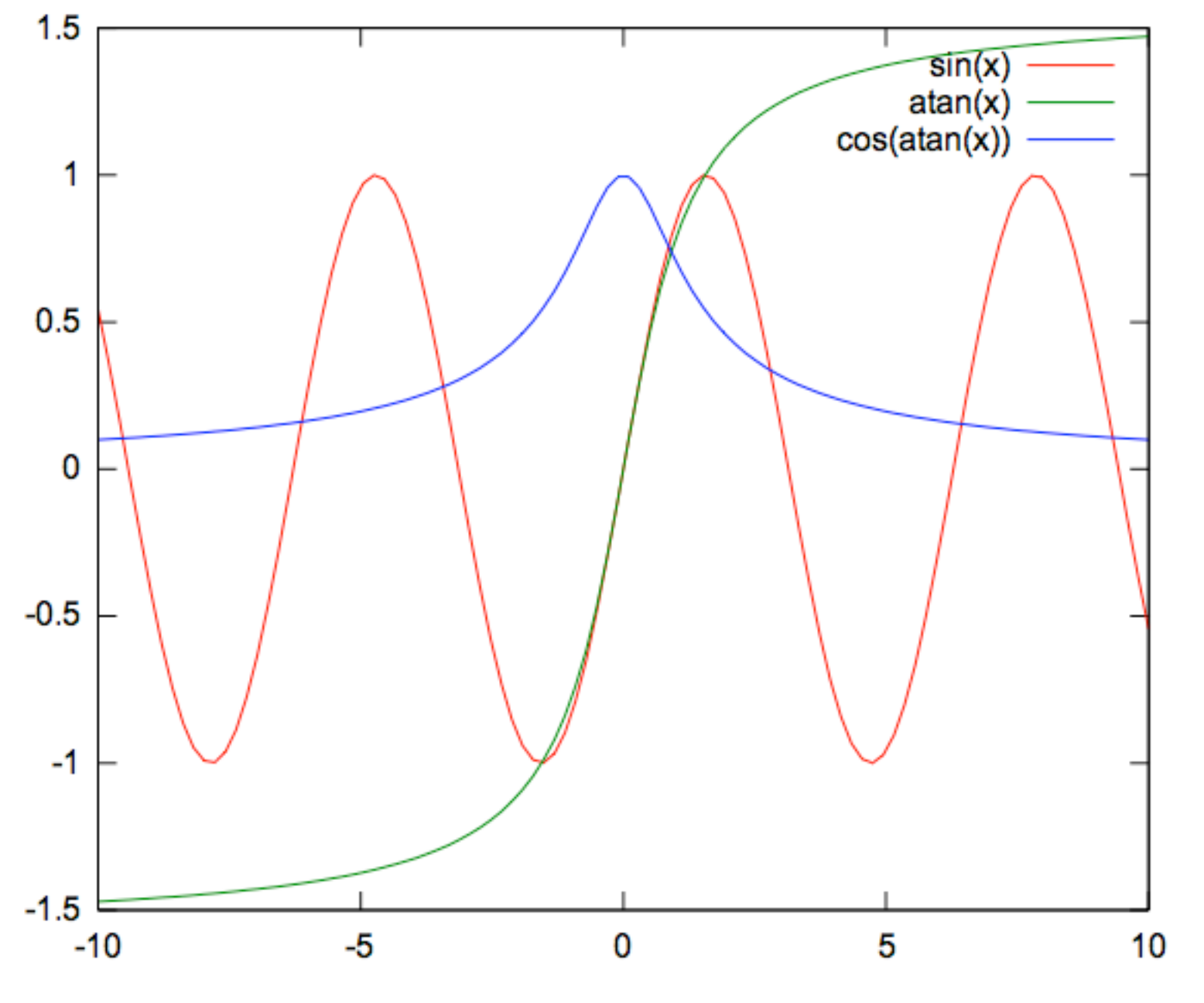

JFreeChart : HistogramDemo1
Histogram Demo 1
Scatter Plot
(sel mvn-samp :columns 1)
Histogram
Density
Scatter Plot Demo 1
Scatter Plot Demo 1
Y
X
Sample 0
Sample 1
Sample 2
Sample 3

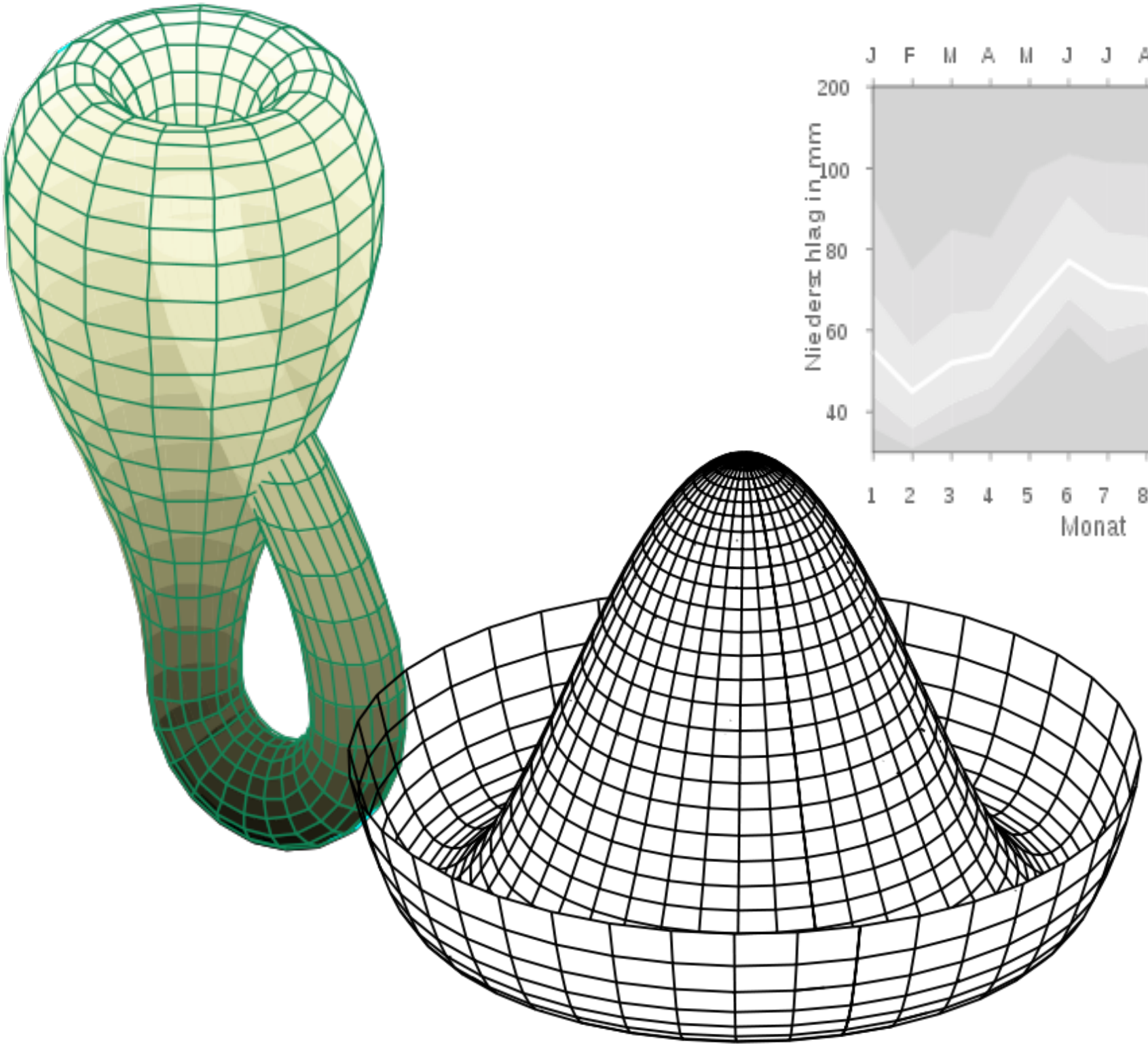
# GNUPLOT

- A very long history
- Stable data analysis and plotting libraries
- Active development (HTML5 Canvas, 3D plotting, animation etc.)
- Limited bindings to Ruby (much easier to use with IO.popen)


Gnuplot
IN ACTION
Understanding data with graphs
Philipp K. Janert

```
def gnuplot(commands)
  IO.popen("gnuplot", "w") {|io| io.puts command
end

# Create an SVG plot using Gnuplot's
# built-in mathematical functions
commands =
  %Q(
     set terminal svg
     set output "curves.svg"
     plot [-10:10] sin(x),atan(x),cos(atan(x))
    )
gnuplot(commands)
```


1.5
1
0.5
0
-0.5
-1
-1.5
-10
-5
0
5
10
sin(x)
atan(x)
cos(atan(x))

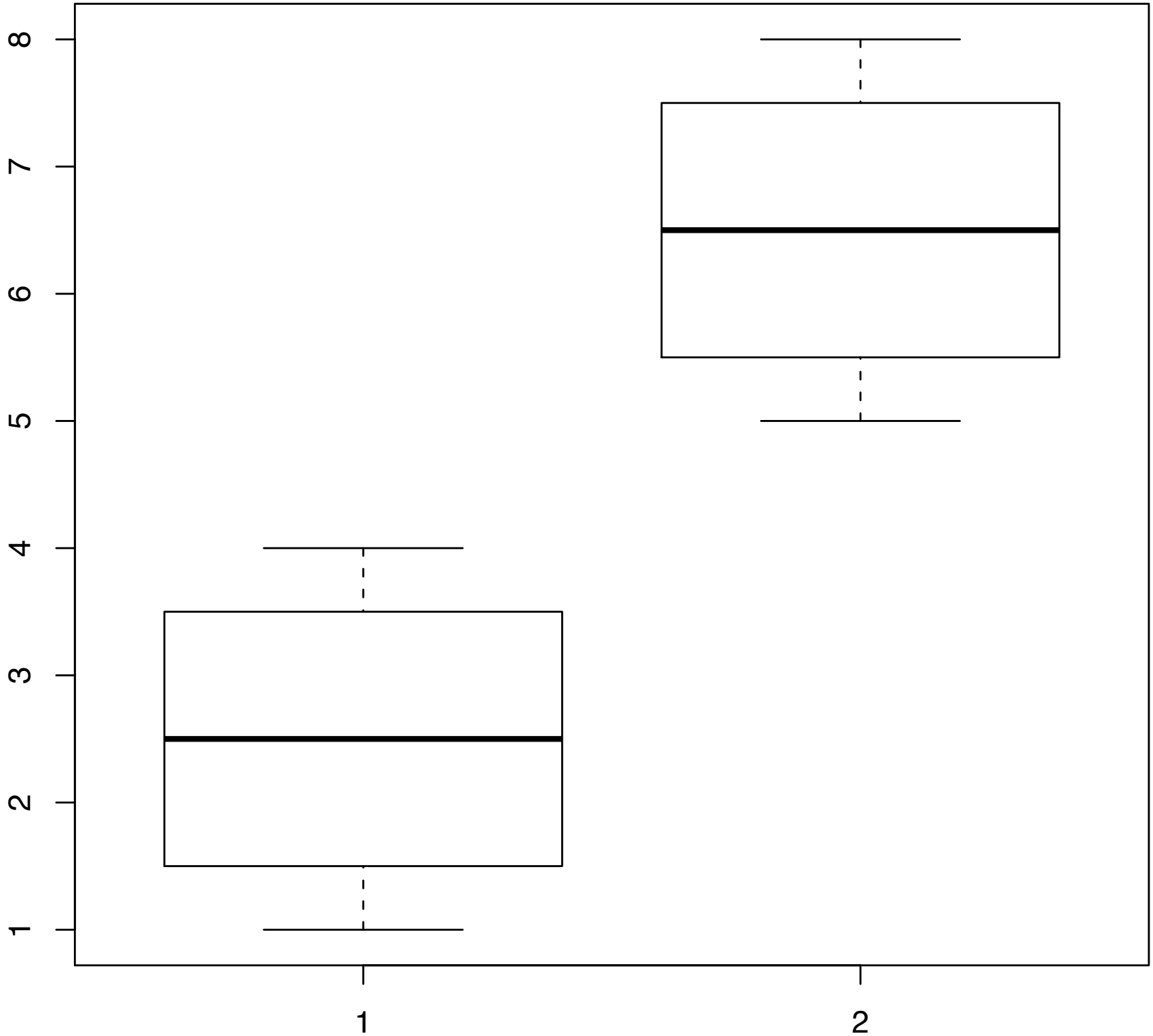

J F M A M J J A S O N D
200
100
80
60
40
Niederschlag in mm
10%
15%
50%
15%
10%
1 2 3 4 5 6 7 8 9 10 11 12
Monat

R

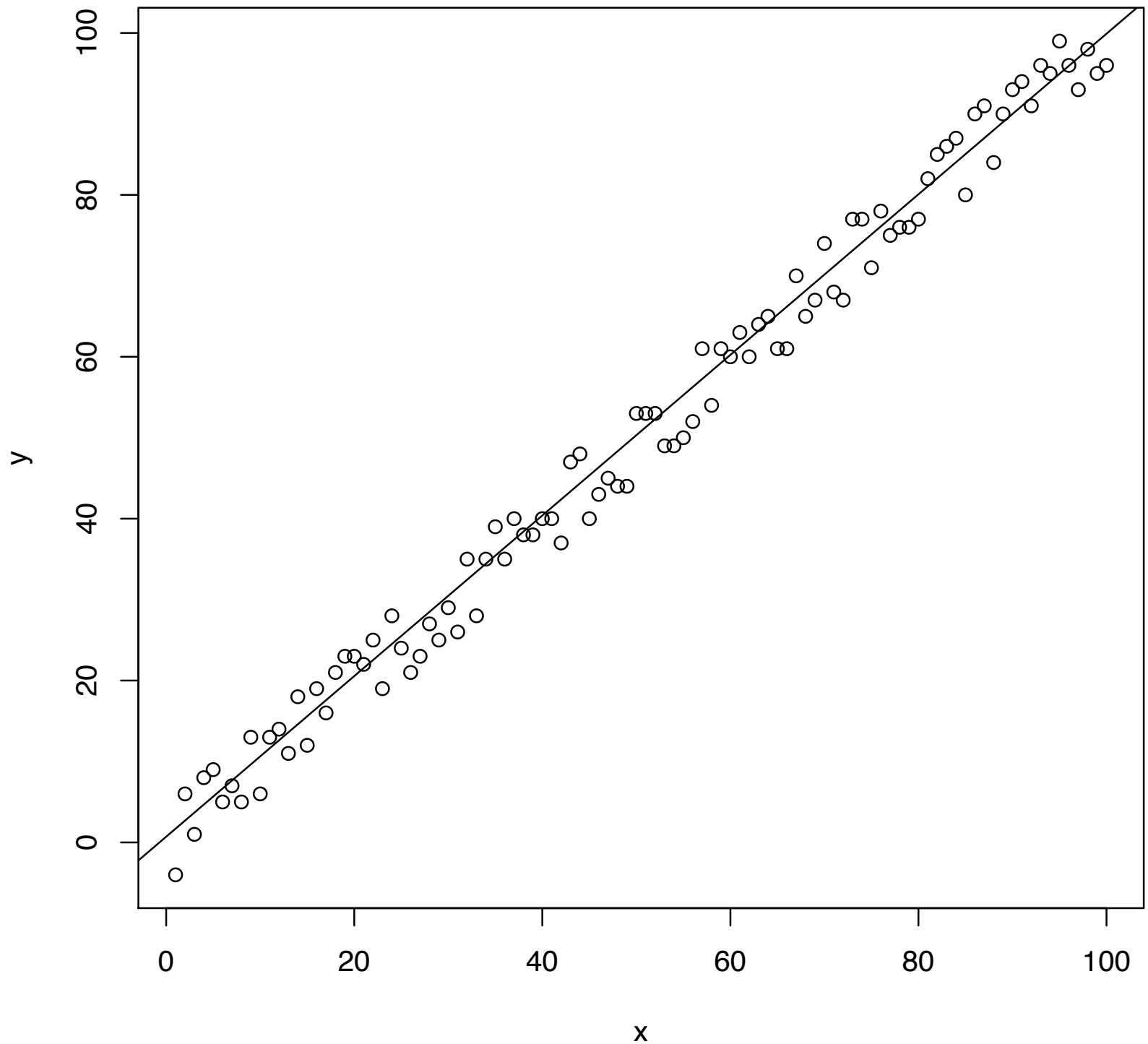
- Plotting and much more ...
- Data analysis, linear algebra, modeling, financial, classical statistical tests, time-series analysis, classification, clustering ...
- Two ways to use:

```
require 'rsruby'

# Simple way of generating plots with R is to provide R
# directly to the RSRuby instance.
cmd = %Q(
  pdf(file = "r_directly.pdf")
  boxplot(c(1,2,3,4),c(5,6,7,8))
  dev.off()
)

r = RSRuby.instance
r.eval_R(cmd)
```

```
r2 = RSRuby.instance
r2.pdf('rsruby.pdf')
r2.boxplot([1,2,3,4],[5,6,7,8])
r2.dev_off.call
```


8
7
6
5
4
3
2
1
1
2

```ruby
# -- A more complex example --
# Fitting a line to some data
r.pdf('line_fit.pdf')

# Define our data using ruby arrays
x = (1..100).to_a
y = x.map{|xi| xi + (rand(10)-5)}

# to process using R commands we must assign to R variables
r.assign('x', x)
r.assign('y', y)

# Fit a linear model to the data
fit = r.lm('x ~ y')

# .. and plot
r.plot(x,y,:xlab => "x", :ylab => "y")
r.abline(fit["coefficients"]["(Intercept)"], fit["coefficients"]["y"])
r.dev_off.call
```


100
80
60
40
20
0
y
0
20
40
60
80
100
x

# Questions?