St. Louis
Clojure

Clojure Incanter

Christopher Mark Gore

cgore.com

Tuesday, April 28, AD 2015

Why Incanter?

- charts
- statistics
- data
- graphics
- don’t have to use R or MATLAB!

Getting Started: Your `project.clj`

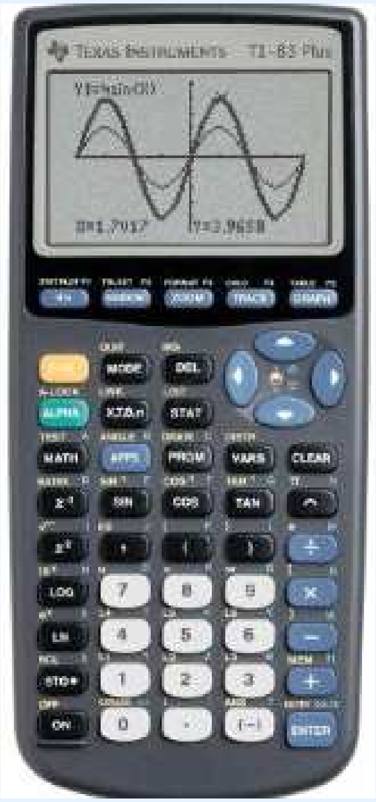
```
:dependencies [... [incanter "1.5.6"] ...]
```

Getting Started: Your Namespace Declaration

```
(ns code.core
  "Howdy␣Incanter!"
  (:require [incanter.core :as i
             :refer [$ $order $rollup $where conj-cols
                     conj-rows dataset dim save
                     to-dataset view]]
            [incanter.datasets :as ids]
            [incanter.stats :as is]
            [incanter.charts :as ic]
            [incanter.io :as iio
                         :refer [read-dataset]]))
```

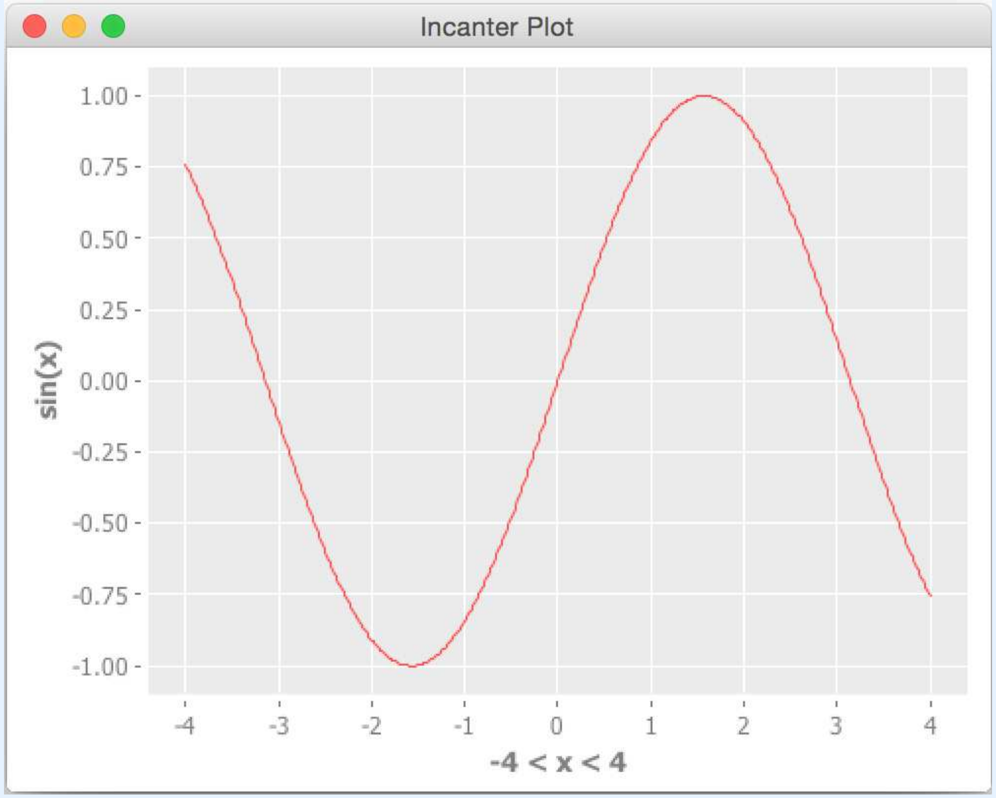
Sine Waves

The first thing I try to do with any plotting system is a simple sine wave. If a plotting library can't easily do that, it's outclassed by a cheap calculator for junior-high school students.



Sine Waves

```
(view (ic/function-plot #(Math/sin %) -4 4
                        :y-label "sin(x)"))
```



Data Sets

You probably want to look at data if you are interested in Incanter. For a really small data set, you might just define it inline.

```
(def small-data (dataset ["x" "y" "theta"]
                          [[1   2   3]
                           [4   5   6]
                           [7   8   9]]))
```

Data Sets from CSVs

If you are working with a real data set, then it's probably living in a CSV file or a database.

```
(def pass-data (read-dataset "../Pass.csv"
                             :header true))
(def fail-data (read-dataset "../Pass.csv"
                             :header true))
```

Data Sets from Hash Maps

Clojure *loves* hash maps. How do you make a data set out of them?

```
(def data-from-hashmaps (to-dataset [{:x 1 :y 2}
                                     {:x 3 :y 4}
                                     {:x 5 :y 6}]))
```

Data Sets from Vectors

```
(def data-from-vecs (to-dataset [[1 2 3]
                                 [4 5 6]
                                 [7 8 9]]))
(def data-cols (conj-cols [1 4 7]
                          [2 5 8]
                          [3 6 9]))
(def data-rows (conj-rows [1 2 3]
                          [4 5 6]
                          [7 8 9]))
```

Data Sets from the Internet

There's no need to download the CSV, if you know the path to it.

```
(def air-passengers
  (read-dataset
   (str "http://vincentarelbundock.github.io"
        "/Rdatasets/csv/datasets/AirPassengers.csv")
   :header true))
```

Included Sample Data Sets

Incanter has a lot of sample data sets included, mostly borrowed from R. Standard data sets are commonly used if you need to test out an algorithm, or compare it to existing algorithms.

```
(def hec (ids/get-dataset :hair-eye-color))
```

$$\begin{pmatrix} :hair & :eye & :gender & :count \\ black & brown & male & 32 \\ black & blue & male & 11 \\ \vdots & \vdots & \vdots & \vdots \end{pmatrix}$$

Saving Data Sets

Incanter provides an easy way to save your data sets to CSV files for use in other tools.

```
(save some-data "some.csv")
```

The $ Operator

The $ operator is a shortcut to get that column of data out of a dataset.

```
(defn x [dataset]
  ($ :x dataset))
(defn y [dataset]
  ($ :y dataset))
(defn theta [dataset]
  ($ :theta dataset))
(defn mpi [dataset]
  ($ (keyword "Monthly␣Personal␣Income") dataset))
```

Multiple Columns with the `$` Operator

To select a few columns:

```
($ ["x" "y"] small-data)
```

To remove one of the columns:

```
($ [:not "theta"] small-data)
```

Both produce:

$$\begin{pmatrix} x & y \\ 1 & 2 \\ 4 & 5 \\ 7 & 8 \end{pmatrix}$$

Single Rows with the $ Operator

We can select a few columns:

```
($ ["x" "y"] small-data)
```

$$\begin{pmatrix} x & y \\ 1 & 2 \\ 4 & 5 \\ 7 & 8 \end{pmatrix}$$

And then select a single row, zero-indexed:

```
($ 1 ["x" "y"] small-data) ; Returns '(4 5)
```

The `$where` Operator

```
($where {:hair "red"} hec) ; Only with red hair
```

$$\begin{pmatrix}
:hair & :eye & :gender & :count \\
red & brown & male & 10 \\
red & blue & male & 10 \\
red & hazel & male & 7 \\
red & green & male & 7 \\
red & brown & female & 16 \\
red & blue & female & 7 \\
red & hazel & female & 7 \\
red & green & female & 7
\end{pmatrix}$$

The `$where` Operator

```
($where {:count {:lt 5}} hec) ; only small samples
```

$$\begin{pmatrix} :hair & :eye & :gender & :count \\ black & green & male & 3 \\ blond & brown & male & 3 \\ black & green & female & 2 \\ blond & brown & female & 4 \end{pmatrix}$$

The `$where` Operator

```
($where (fn [row] ; We can do any function we want.
          (and (= (row :hair) "blond")
               (= (row :eye) "blue")))
        hec)
```

$$\begin{pmatrix} :hair & :eye & :gender & :count \\ blond & blue & male & 30 \\ blond & blue & female & 64 \end{pmatrix}$$

The `$order` Operator

```
($order :count :desc hec)
```

$$\begin{pmatrix} :hair & :eye & :gender & :count \\ brown & brown & female & 66 \\ blond & blue & female & 64 \\ brown & brown & male & 53 \\ brown & blue & male & 50 \\ black & brown & female & 36 \\ brown & blue & female & 34 \\ black & brown & male & 32 \\ blond & blue & male & 30 \\ \vdots & \vdots & \vdots & \vdots \end{pmatrix}$$

The `$rollup` Operator

```
($rollup i/sum :count [:hair :eye] hec)
```

$$\begin{pmatrix}
:eye & :hair & :count \\
hazel & brown & 54.0 \\
brown & blond & 7.0 \\
green & red & 14.0 \\
brown & red & 26.0 \\
hazel & red & 14.0 \\
blue & red & 17.0 \\
blue & blond & 94.0 \\
green & black & 5.0 \\
\vdots & \vdots & \vdots
\end{pmatrix}$$

Combining Operators

```
($order :count :desc
        ($rollup i/sum :count [:hair :eye] hec))
```

$$\begin{pmatrix} :eye & :hair & :count \\ brown & brown & 119.0 \\ blue & blond & 94.0 \\ blue & brown & 84.0 \\ brown & black & 68.0 \\ hazel & brown & 54.0 \\ green & brown & 29.0 \\ \vdots & \vdots & \vdots \end{pmatrix}$$

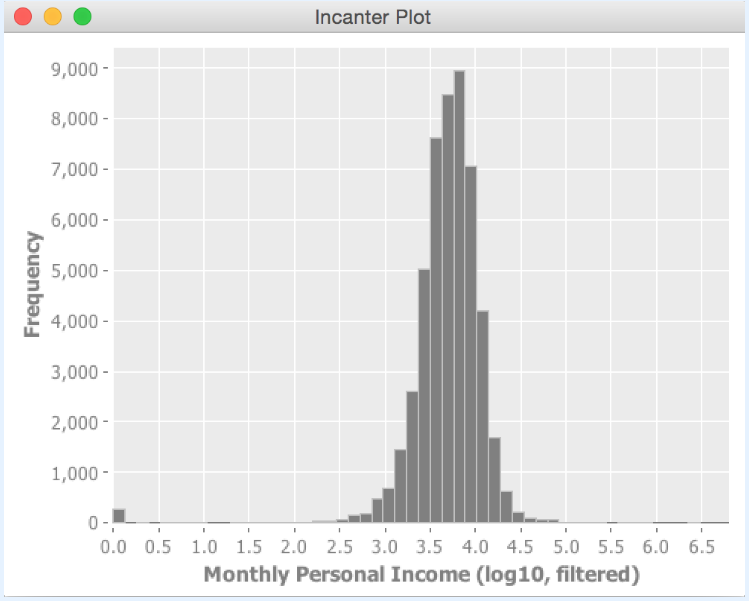
Statistics

There is a lot of statistics available. Some of the basics:

```
(def mpi-stats {:mean (is/mean mpi-filtered)
                :variance (is/variance mpi-filtered)
                :std-dev (is/sd mpi-filtered)
                :median (is/median mpi-filtered)
                :kurtosis (is/kurtosis mpi-filtered)})
```

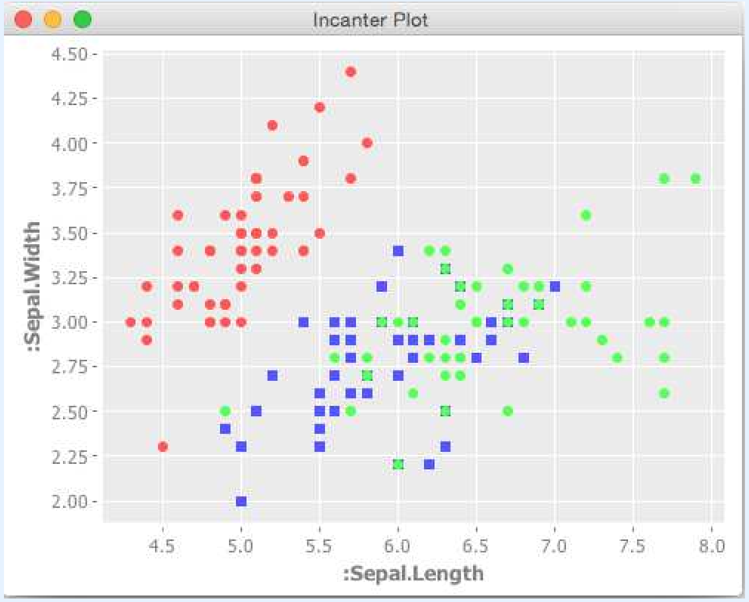
Histograms

```
(let [mpi-filtered (filter #(< 0 %) (mpi pass-data))
      mpi-log10 (map #(Math/log10 %) mpi-filtered)]
  (view (ic/histogram mpi-log10
                      :x-label "Monthly␣Personal␣Income"
                      :nbins 50)))
```



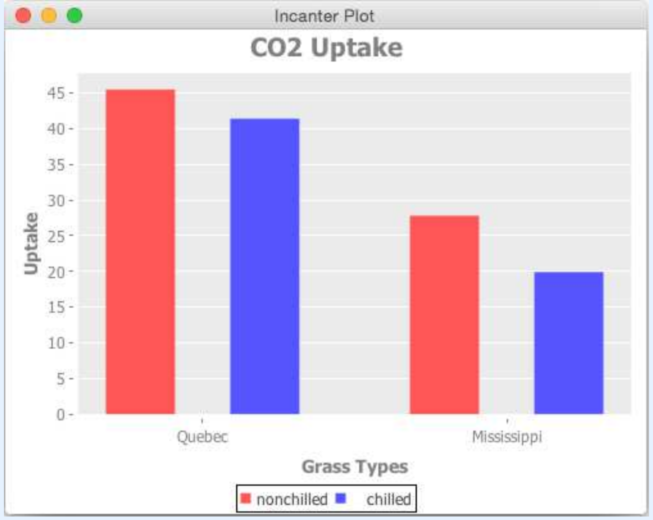
Scatter Plots

```
(i/with-data (ids/get-dataset :iris)
  (view (ic/scatter-plot :Sepal.Length :Sepal.Width
                         :group-by :Species)))
```



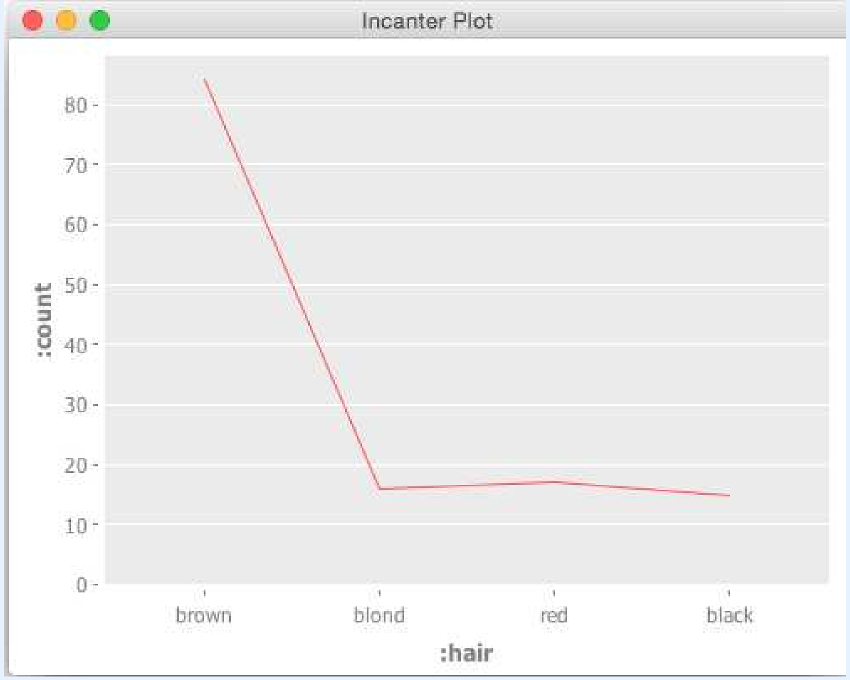
Bar Charts

```
(i/with-data (ids/get-dataset :co2)
  (view (ic/bar-chart :Type :uptake
                      :title "CO2␣Uptake"
                      :group-by :Treatment
                      :x-label "Grass␣Types"
                      :y-label "Uptake"
                      :legend true)))
```



Line Charts

```
(i/with-data ($rollup i/sum :count [:hair :eye] hec)
  (view (ic/line-chart :hair :count)))
```



And there’s a lot more I didn’t mention!

Questions?